Sensu Go

Contents

- InfluxDB
- Jira
- OpenTSDB
- PagerDuty
- Prometheus
- Puppet
- Rundeck
- SaltStack
- ServiceNow
- Slack
- Sumo Logic
- TimescaleDB
- Wavefront

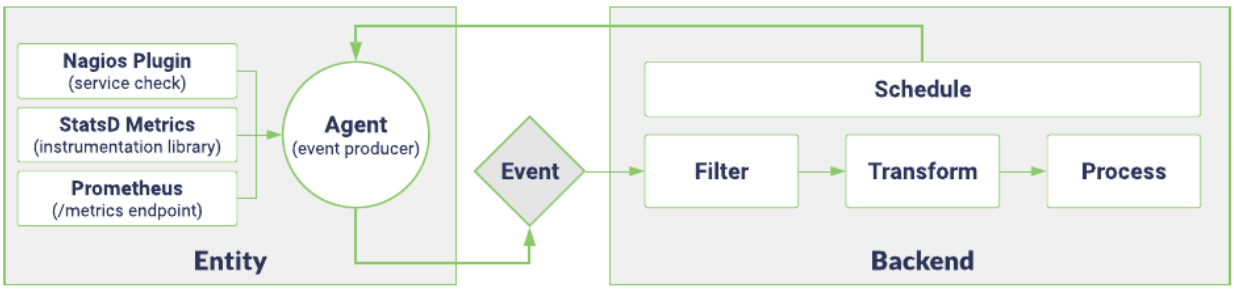
Learn Sensu

- Concepts and Terminology
- Live Demo

Learn about licensing

Sensu is a complete solution for monitoring and observability at scale.Sensu Go is designed to give you visibility into everything you care about: traditional server closets, containers, applications, the cloud, and more.

Take a tour **or click any element in the Sensu observability pipeline to jump to it.**



Sensu Observability Pipeline

Sensu is an agent-based observability tool that you install on your organization's infrastructure. The Sensu backend gives you a flexible, automated pipeline to filter, transform, and process alerts and metrics.

Sensu Go is operator-focused and developer-friendly and integrates with popular monitoring and observability tools.Deploy Sensu Go for on-premises and public cloud infrastructures, containers, bare metal, or any other environment.

Get started now and feel the #monitoringlove.

Filtered, context-rich alerts that improve incident response

Get meaningful alerts when and where you need them so you can reduce alert fatigue and speed up incident response. Sensu gives you full control over your alerts with flexible event filters, check hooks for context-rich notifications, reporting, observation data handling, and auto-remediation.

Extend functionality and integrate with existing workflows with the Sensu Catalog

Use the Sensu Catalog, the online marketplace for monitoring and observability integrations, to find and install integrations directly in your browser.

Sensu's open architecture integrates with the tools and services you already use, like Ansible, Amazon EC2, InfluxDB, Kubernetes, PagerDuty, Saltstack, and Sumo Logic. The Sensu Catalog also includes standard system checks and metrics collectors.

To start integrating Sensu with your existing workflows, read the Sensu Catalog documentation, check out our featured integrations, search for plugins in Bonsai, the Sensu asset hub, or write your own Sensu plugins in any language.

Automate with agent registration-deregistration and check subscriptions

Sensu agents automatically register and deregister themselves with the Sensu backend so you can collect observation data about ephemeral infrastructure without getting overloaded with alerts.

Instead of setting up traditional one-to-one entity-to-check mapping, use Sensu's subscriptions to make sure your entities automatically run the appropriate checks for their functionality.

Built-in support for industry-standard tools

Know what's going on everywhere in your system. Sensu supports industry-standard metric formats like Nagios performance data, Graphite plaintext protocol, InfluxDB line protocol, OpenTSDB data

specification, Prometheus Exposition Text Format, and StatsD metrics. Use the Sensu agent to collect metrics alongside check results, then use the Sensu observability pipeline to route observation data to a time-series database like InfluxDB.

Intuitive API with command line and web interfaces

The Sensu API and the `sensuctl` command-line tool allow you (and your internal customers) to create checks, register entities, manage configuration, and more. The Sensu web UI provides a unified view of your entities, checks, and events, as well as a user-friendly silencing tool.

Commercial software based on open core

Sensu Go is a commercial product, based on an open source core that is freely available under a permissive MIT License and publicly available on GitHub. Learn about our commercial support packages and features designed for observability at scale.

Sensu Go is the latest version of Sensu, designed to be portable, straightforward to deploy, and friendly to containerized and ephemeral environments. Sensu Inc. released Sensu Go OSS as open source in 2017, and it is now a part of Sumo Logic Inc. (SUMO).

Sensu is a comprehensive monitoring and observability solution for enterprises, providing complete visibility across every system, every protocol, every time — from Kubernetes to bare metal.

Sensu Go release notes

- 6.4.0 release notes
- 6.3.0 release notes
- 6.2.7 release notes
- 6.2.6 release notes
- 6.2.5 release notes
- 6.2.4 release notes
- 6.2.3 release notes
- 6.2.2 release notes
- 6.2.1 release notes
- 6.2.0 release notes
- 6.1.4 release notes
- 6.1.3 release notes
- 6.1.2 release notes
- 6.1.1 release notes
- 6.1.0 release notes
- 6.0.0 release notes
- 5.21.5 release notes
- 5.21.4 release notes
- 5.21.3 release notes
- 5.21.2 release notes
- 5.21.1 release notes
- 5.21.0 release notes
- 5.20.2 release notes
- 5.20.1 release notes
- 5.20.0 release notes
- 5.19.3 release notes
- 5.19.2 release notes
- 5.19.1 release notes
- 5.19.0 release notes

- 5.18.1 release notes
- 5.18.0 release notes
- 5.17.2 release notes
- 5.17.1 release notes
- 5.17.0 release notes
- 5.16.1 release notes
- 5.16.0 release notes
- 5.15.0 release notes
- 5.14.2 release notes
- 5.14.1 release notes
- 5.14.0 release notes
- 5.13.2 release notes
- 5.13.1 release notes
- 5.13.0 release notes
- 5.12.0 release notes
- 5.11.1 release notes
- 5.11.0 release notes
- 5.10.2 release notes
- 5.10.1 release notes
- 5.10.0 release notes
- 5.9.0 release notes
- 5.8.0 release notes
- 5.7.0 release notes
- 5.6.0 release notes
- 5.5.1 release notes
- 5.5.0 release notes
- 5.4.0 release notes
- 5.3.0 release notes
-

- 5.2.1 release notes
- 5.2.0 release notes
- 5.1.1 release notes
- 5.1.0 release notes
- 5.0.1 release notes
- 5.0.0 release notes

Versioning

Sensu Go adheres to semantic versioning using MAJOR.MINOR.PATCH release numbers, starting at 5.0.0. MAJOR version changes indicate incompatible API changes. MINOR versions add backward-compatible functionality. PATCH versions include backward-compatible bug fixes.

Upgrading

Read the upgrade guide for information about upgrading to the latest version of Sensu Go.

6.8.2 release notes

October 6, 2022 — The latest release of Sensu Go, version 6.8.2, is now available for download.

Sensu Go 6.8.2 includes logging improvements with the addition of check names for failed check execution requests. We also added a label to events with a truncated check output and now automatically restart the agent on Windows platforms after failures. The 6.8.2 patch release also modifies the keepalive startup logic and fixes a number of web UI issues in the Entities and configuration resource pages.

Read the upgrade guide to upgrade Sensu to version 6.8.2.

IMPROVEMENTS:

- When check output is truncated due to the max_output_size configuration, the events the check produces will include a `sensu.io/output_truncated_bytes` label.
- Agent log messages now include the check name when a check execution request fails.

- On Windows platforms, the Sensu Agent service now automatically restarts after failures.

FIXES:

- (Commercial feature) In the web UI, restored the silence function on the Entities page.
- (Commercial feature) In the web UI, resource pages now automatically refresh after creating resources.
- (Commercial feature) The web UI now displays pipeline definitions under the **RAW** tab on individual pipeline resource pages.
- (Commercial feature) In the web UI, corrected the link to the entity reference in the Edit Entity modal.
- (Commercial feature) In the web UI, errors displayed when deleting and re-adding an asset from asset page have been addressed.
- (Commercial feature) In the web UI, fixed the validation for resource names and array fields to prevent crashes.
- (Commercial feature) In the web UI, the configuration resource pages now show an empty list instead of an endless loading indicator for users who do not have the required permissions.
- (Commercial feature) In the web UI, fixed a bug that could cause a crash when an authorized user does not have an explicitly set username.
- (Commercial feature) In the web UI, temporarily disabled saved searches on Entity, Services, Silences, and Check pages.
- (Commercial feature) In the web UI, fixed a bug that prevented individual resource pages from displaying annotations and labels on initial page load.
- (Commercial feature) In the web UI, when users do not have the required permissions to perform a specific action, the action's button is now disabled with a tooltip to explain the reason.
- Modified keepalive startup so that etcd lease errors will not cause sensu-backend crashes.

6.8.1 release notes

September 13, 2022 — The latest release of Sensu Go, version 6.8.1, is now available for download.

Sensu Go 6.8.1 includes web UI fixes for OIDC refresh token expiry and information displayed on the Entities page, as well as a change in how check subdue begin and end times are evaluated.

Read the upgrade guide to upgrade Sensu to version 6.8.1.

FIXES:

- (Commercial feature) In the web UI, OIDC refresh token requests now properly invoke the sign-in dialog instead of causing an `HTTP 404 Not Found` error.
- (Commercial feature) In the web UI, the entity list no longer displays the values of redacted labels.
- Check subdues are now evaluated as half-open intervals so that they are inclusive of the begin time and +1-second exclusive of the end time. Previously, subdue periods were evaluated as closed intervals and were exclusive of the begin and end times. This change prevents unintended gaps between subdues.

6.8.0 release notes

August 29, 2022 — The latest release of Sensu Go, version 6.8.0, is now available for download.

Sensu Go 6.8.0 delivers a mix of new features, valuable improvements, and helpful fixes. The new /ready API endpoint provides information about backend readiness, and the api-serve-wait-time and agent-serve-wait-time backend configuration options can help prevent instability during sensu-backend startup. The web UI now includes dedicated resource pages for assets, pipelines, role-based access control (RBAC) resources, and secrets. Plus, the resource pages now include details that give you more information about your resources at a glance. We've also fixed bugs that could cause backend crashes or result in incorrect event.check.issued values and improved prioritization to prevent keepalive event creation storms.

Read the upgrade guide to upgrade Sensu to version 6.8.0.

NEW FEATURES:

- Added the api-serve-wait-time and agent-serve-wait-time backend configuration options. Use api-serve-wait-time to delay serving API requests and agent-serve-wait-time to delay accepting agent connections after starting the backend.
- Added the /ready API endpoint to provide HTTP GET access to information about whether a Sensu instance is ready to serve API requests and accept agent connections.

IMPROVEMENTS:

- (Commercial feature) The web UI now includes resource pages for assets, pipelines, role-

based access control (RBAC) resources, and secrets.

- (Commercial feature) In the web UI, resource pages now render resources in an infinite list, with a total row count provided at the bottom-right of the list.
- (Commercial feature) The resource pages in the web UI now display additional information about each resource, such as subscriptions, labels, API versions, and command attribute values.
- (Commercial feature) In the web UI, the system information modal now displays the name of the connected Sensu backend.
- Eventd now prioritizes keepalive events over other events to help prevent keepalive event creation storms and mass agent disconnects.

FIXES:

- (Commercial feature) When the event.check.issued attribute has a null value, the event detail page in the web UI now displays `N/A` instead of `December 31, 1969`.
- Fixed a bug that could cause a sensu-backend crash if the BackendIDGetter encountered etcd client unavailability.

6.7.5 release notes

August 10, 2022 — The latest release of Sensu Go, version 6.7.5, is now available for download.

Sensu Go 6.7.5 upgrades the graphql-go/graphql library to address CVE-2022-37315 in which a malicious actor may craft a query that can crash the backend instance.

Read the upgrade guide to upgrade Sensu to version 6.7.5.

IMPROVEMENTS

- Upgraded graphql-go/graphql to remediate CVE-2022-37315.

6.7.4 release notes

July 15, 2022 — The latest release of Sensu Go, version 6.7.4, is now available for download.

Sensu Go 6.7.4 upgrades the Go version to 1.17.12.

Read the upgrade guide to upgrade Sensu to version 6.7.4.

IMPROVEMENTS

- Upgraded Go version from 1.17.6 to 1.17.12.

6.7.3 release notes

July 7, 2022 — The latest release of Sensu Go, version 6.7.3, is now available for download.

Sensu Go 6.7.3 includes fixes for the Sensu Catalog sort order and web UI notifications, as well a database issue that could cause backends to crash. We also fixed bugs that affected business service monitoring (BSM) service components and removed limits on Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) searches. This patch release includes a change in how agents execute check requests to prevent `check execution still in progress` failures.

Read the upgrade guide to upgrade Sensu to version 6.7.3.

IMPROVEMENTS

- (Commercial feature) Added supported packages for the Sensu backend, Sensu agent, and sensuctl for RHEL 9.

FIXES

- (Commercial feature) When using the business service monitoring (BSM) feature, service component metadata is now included in the `check` scope of events the service component generates. Also fixed a bug that could cause BSM service component queries to retrieve events that do not match the specified query expressions.
- (Commercial feature) Removed a database constraint that could cause backends to crash when running agents on hosts that have many addresses associated with a single network interface.
- (Commercial feature) Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) searches are no longer limited to 1000 results.
- (Commercial feature) In the web UI, Sensu Catalog integrations are now listed alphabetically.
- (Commercial feature) In the web UI's automated Sensu Plus setup dialog, the value in the **Source URL** field is no longer truncated.
- (Commercial feature) In the web UI, pop-up notifications at the bottom of the page are no

longer obscured by other content.

- To prevent `check execution still in progress` failures, agents will no longer execute check requests with issued timestamps that are equal to or older than the issued timestamp for the last executed check request with the same check name.

6.7.2 release notes

May 12, 2022 — The latest release of Sensu Go, version 6.7.2, is now available for download.

Sensu Go 6.7.2 includes a fix for sensu-backend stability and adds an active poller for PostgreSQL config changes. We've also improved the Sensu Plus modal in the web UI.

Read the upgrade guide to upgrade Sensu to version 6.7.2.

IMPROVEMENTS

- (Commercial feature) In the web UI, the Sensu Plus modal dialog now directs users who already have a Sumo Logic account to follow the instructions to manually set up Sensu Plus.
- (Commercial feature) In the web UI, the Sensu Plus post-setup modal dialog now indicates success when you use the Copy button to copy the Source URL.
- (Commercial feature) Added supported packages for the Sensu backend, Sensu agent, and sensuctl for Ubuntu 22.04.
- Added the etcd-unsafe-no-fsync backend configuration option, which makes it possible to run sensu-backend with an embedded etcd node for testing and development without placing too much load on the file system.
- Upgraded etcd version from 3.5.2 to 3.5.4.

FIXES

- (Commercial feature) Fixed a bug that could cause a backend crash when pruning SumoLogicMetricsHandler and TCPStreamHandler resource types.
- (Commercial feature) Implemented an active poller for PostgreSQL configuration changes.
- The correct round robin scheduler source ("etcd" or "postgres") is now printed in events and logs.

6.7.1 release notes

April 28, 2022 — The latest release of Sensu Go, version 6.7.1, is now available for download.

Sensu Go 6.7.1 delivers several improvements and fixes in the Sensu Catalog, along with an update to cron scheduler logging. We've also included fixes for data races in schedulerd and agentd.

Read the upgrade guide to upgrade Sensu to version 6.7.1.

IMPROVEMENTS

- (Commercial feature) Updated install button styling and improved text padding and margins for integration details and configuration dialog icons in the Sensu Catalog.
- (Commercial feature) In the Sensu Catalog, the integrations list is now sorted alphabetically.
- The cron scheduler now logs that it is stopping before it begins the process of stopping.

FIXES

- (Commercial feature) In the Sensu Catalog, fixed an issue that prevented the overwrite function from properly overwriting existing resources.
- Fixed several data races in schedulerd.
- Mitigated a data race in agentd sessions.

6.7.0 release notes

April 21, 2022 — The latest release of Sensu Go, version 6.7.0, is now available for download.

Sensu Go 6.7.0 includes a number of new features, improvements and fixes, including Sensu Go's newest feature, the Sensu Catalog. The Catalog is a marketplace within the Sensu web UI that facilitates new user onboarding and deploying production-ready monitoring in minutes. Sensu Go 6.7.0 also includes metric threshold evaluation, keepalive pipelines, and check subdues. We've improved the onboarding workflow for Sensu Plus so you can seamlessly transmit Sensu observability data to Sumo Logic, added support for arrays of strings and objects in the `sensu.CheckDependencies` Sensu query expression, added an attribute to the GlobalConfig specification, and more. Bug fixes in Sensu Go 6.7.0 include adding pipelines within the event.check object, correcting TCP stream handler `max_connections` behavior, and detecting ARM platform accurately.

Read the upgrade guide to upgrade Sensu to version 6.7.0.

NEW FEATURES:

- (Commercial feature) Added the Sensu Catalog, an online marketplace for monitoring and observability integrations that allows you to find, configure, and install integrations directly from the Sensu web UI.
- Added metric threshold evaluation to provide real-time alerts based on the metrics your Sensu checks collect.
- Added the `keepalive-pipelines` agent configuration option, which allows you to specify pipelines for processing keepalive events.
- Added the check `subdues` attribute, which you can use to schedule alert-free periods of time directly in check definitions.

IMPROVEMENTS:

- (Commercial feature) For Sensu Plus setup, Sensu now automatically creates a Sumo Logic account and configures an HTTP Logs & Metrics Source for customers who start the process from the Sensu web UI.
- (Commercial feature) Markdown formatting is now supported for the signin_message attribute value in the GlobalConfig specification.
- (Commercial feature) Added the serialization_format attribute to the GlobalConfig specification, which you can use to specify the default format for resource definitions in the web UI (YAML or JSON).
- (Commercial feature) Added the license_expiry_reminder attribute to the GlobalConfig specification, which you can use to specify the number of days before license expiration to begin displaying the license expiration banner in the web UI.
- (Commercial feature) Business service monitoring (BSM) now uses the PostgreSQL round robin Ring V2 implementation, even if the `enable_round_robin` attribute is set to `false` in the PostgresConfig definition.
- (Commercial feature) Added the `sensu_go_etcd_cluster_leases` metric to the backend metrics log to track the count of current etcd leases for debugging.
- (Commercial feature) Added logging for TCP stream handler events.
- The `sensu.CheckDependencies` Sensu query expression now supports arrays of strings and arrays of objects.
- On backend startup, Sensu now creates the `sensu-system` namespace and a backend entity to log secrets provider errors and help prevent spamming the event bus with backend events.
- For connections with faulty TLS configurations, error log entries now include a `source`

property that lists the corresponding agent's IP address and port to identify which agent generated each log entry for troubleshooting.

- Increased the default values for the backend configuration options etcd-election-timeout (from 1000 to 3000) and etcd-heartbeat-interval (from 100 to 300).
- Upgraded etcd version from `3.5.0` to `3.5.2`.

FIXES:

- (Commercial feature) Fixed a bug that could cause pipeline resources to hang when using a TCP stream handler whose `max_connections` attribute is set to greater than zero.
- In events, event.check objects now include any pipelines specified in the check configuration.
- Socket handlers that are interrupted by an error mid-write will no longer cause a sensu-backend panic. Also, socket handlers will now respect their timeout settings after the initial connection is established.
- Fixed a bug that prevented accurate ARM version detection for sensu-agent.

6.6.6 release notes

February 16, 2022 — The latest release of Sensu Go, version 6.6.6, is now available for download.

Sensu Go 6.6.6 includes several web UI fixes for GraphQL queries. This patch release also contains fixes for the PostgreSQL event store, including improving retry logic when the event store is unavailable, as well as not reverting to etcd as a fallback event store.

Read the upgrade guide to upgrade Sensu to version 6.6.6.

IMPROVEMENTS

- (Commercial feature) In the web UI, added error type to GraphQL metrics to help track down slow queries.

FIXES

- (Commercial feature) When the PostgreSQL provider is configured with "strict: true", the provider will attempt to connect to an unavailable PostgreSQL server indefinitely instead of reverting to etcd as an event store after three failed connection attempts.
- (Commercial feature) When the PostgreSQL provider is configured to use strict mode, the provider confirms whether the current user has `CREATE` privileges within the current schema,

not the current database.

- (Commercial feature) The PostgreSQL provider now respects context cancellation and will fail immediately when users issue a termination signal.
- (Commercial feature) Fixed an issue where metrics would not be recorded when an error occurred.
- (Commercial feature) In the web UI, fixed an issue with GraphQL queries where an offset of >= 500 couldn't be used when paging through entities.

6.6.5 release notes

February 3, 2022 — The latest release of Sensu Go, version 6.6.5, is now available for download.

Sensu Go 6.6.5 includes several web UI improvements to reduce cluster load and adds a message to clarify web UI search results for the events and entities pages. This patch release also fixes bugs in round robin scheduling and the PostgreSQL configuration watcher and removes outdated language in an interactive-mode prompt for sensu-backend upgrade.

Read the upgrade guide to upgrade Sensu to version 6.6.5.

IMPROVEMENTS

- (Commercial feature) When using PostgreSQL, queries for multiple entity states are now more efficient.
- (Commercial feature) In the web UI, if a search reaches the limit for the events or entities page, the results count at the bottom-right corner of the page now indicates that the total number of matches exceeds the number of results listed.
- (Commercial feature) In the web UI, several changes help reduce cluster load: federated clusters now query remote clusters in parallel; GraphQL resolvers are no longer invoked if the query deadline has already been reached; and we improved the performance of GraphQL queries to the local cluster.

FIXES

- (Commercial feature) Fixed a bug in round robin scheduling that could delay notification routing after creating or updating business service monitoring (BSM) service components.
- (Commercial feature) Fixed a bug in the PostgreSQL configuration watcher that could prevent bsmd from being reenabled after an update.
-

- (Commercial feature) In the web UI, fixed a bug that could result in graphql-go nullification of entity.status values greater than math.MaxInt32.
- Removed Sensu Go 5.x-specific language in the confirmation prompt for sensu-backend upgrade in interactive mode.
- Resolved unpredictable ringv2 behavior when identical subscriptions are created from different contexts.

6.6.4 release notes

January 26, 2022 — The latest release of Sensu Go, version 6.6.4, is now available for download.

Sensu Go 6.6.4 includes a number of bug fixes, security improvements, and a new metric, `sensu_go_event_metric_points_processed`. Fixes in this patch release will help prevent backend crashes when PostgreSQL is taken offline and keep backend entity rows from filling up the entities table. The 6.6.4 patch release also includes several improvements to further secure the web UI.

Read the upgrade guide to upgrade Sensu to version 6.6.4.

IMPROVEMENTS

- (Commercial feature) In the web UI, added the X-Frame-Options header to tell browsers the web application cannot be loaded within an iframe to prevent tailored click-jacking attacks.
- (Commercial feature) In the web UI, added the HSTS header if TLS has been configured to ensure that the browser loads the application and its requisite assets with a secure connection.
- (Commercial feature) In the web UI, added the X-Content-Type-Options nosniff header so that browsers respect the given Content-Type header when loading content referenced by a script tag.
- Added the `sensu_go_event_metric_points_processed` counter metric and included it in Tessen reporting.

FIXES

- (Commercial feature) Fixed bugs in business service monitoring (BSM) and round robin scheduling to prevent missed check executions when PostgreSQL round robin scheduling is enabled.
- (Commercial feature) Fixed a bug that could cause sensu-backend to crash if PostgreSQL was taken offline and restarted.
-

- (Commercial feature) Fixed a bug that could cause ephemeral backend entity rows to fill up the entities table in PostgreSQL.
- (Commercial feature) BSM event selectors can no longer select events outside the service component namespace.
- (Commercial feature) In the web UI, fixed a bug the prevented HTTP requests from being properly cancelled after a context deadline (timeout) was exceeded.
- Fixed a bug that could cause the backend to crash if a pipeline references a non-existent handler.

6.6.3 release notes

December 16, 2021 — The latest release of Sensu Go, version 6.6.3, is now available for download.

Sensu Go 6.6.3 includes improvements to reduce load on clusters and support cluster recovery, as well as a backend configuration option for specifying the internal etcd client log level. Fixes in this patch release will help prevent backend crashes when keepalive leases are revoked and when the backend cannot write to the event log file. In addition, this patch fixes issues that could result in a leaked etcd lease and keep the backend from terminating correctly.

Read the upgrade guide to upgrade Sensu to version 6.6.3.

IMPROVEMENTS

- (Commercial feature) In the web UI, the default polling interval on the entities page is now 30 seconds to help reduce load on clusters. Search results for entities are limited to the first 500 matching entities. Also, the web UI response time and memory usage is substantially improved when opening the entities page in the default state (loading the first page of results, with no search filter applied).
- (Commercial feature) In the web UI, for instances that use etcd for event storage, search results for events are limited to 25,000 matching events.
- Added the `etcd-client-log-level` configuration option for setting the log level of the etcd client used internally within sensu-backend.
- The agentd daemon now starts up after all other daemons, which improves cluster recovery after the loss of a backend.
- When using external etcd (the `no-embed-etcd` backend configuration option is set to `true`), sensu-backend now crashes when its daemons do not stop within 30 seconds, which can happen due to an intentional shutdown or when database unavailability triggers an internal restart. When using embedded etcd, sensu-backend will still try to avoid crashing to prevent

member corruption.

FIXES

- New agent sessions no longer result in a leaked etcd lease.
- sensu-backend now prints a warning and continues instead of crashing when it cannot write to the specified event-log-file.
- Fixed a bug that could cause a crash when keepalive leases are revoked on another backend or by an etcd operator.
- Fixed an issue that could prevent sensu-backend from terminating correctly.
- Proxy entity state is now created when it is missing and a matching entity config already exists.

6.6.2 release notes

December 8, 2021 — The latest release of Sensu Go, version 6.6.2, is now available for download.

The Sensu Go 6.6.2 patch release includes improvements in PostgreSQL health check queries and memory consumption for events and entities pages in the web UI. This release also fixes a web UI issue that provided incorrect links for cluster-wide resources.

Read the upgrade guide to upgrade Sensu to version 6.6.2.

IMPROVEMENTS

- (Commercial feature) Changed the SQL operation for PostgreSQL health check queries to reduce query cost.
- (Commercial feature) In the web UI, removed **Related Entities** from individual pages for events and entities to eliminate the substantial memory consumption required to construct the list.

FIXES

- (Commercial feature) In the web UI, fixed an issue that provided incorrect links for cluster-wide resources. No web UI pages can show events across all namespaces.

6.6.1 release notes

November 29, 2021 — The latest release of Sensu Go, version 6.6.1, is now available for download.

This patch release removes a debugging log entry; adds cron library error information to validation errors for Check and CheckConfig resources; and fixes a web UI bug that expanded the clear silences dialog to the entire frame. In addition, Sensu now sets event timestamps when events are resolved via sensuctl or the web UI.

Read the upgrade guide to upgrade Sensu to version 6.6.1.

IMPROVEMENTS

- To provide additional context, errors returned by the cron library are now included with the errors Sensu returns when validating Check and CheckConfig resources.
- Removed a debugging log entry that could cause logs to grow too big, too quickly.

FIXES

- (Commercial feature) In the web UI, the dialog window for clearing silences no longer expands to the entire frame.
- Sensu now sets event timestamps when you resolve events with sensuctl or in the web UI.

6.6.0 release notes

November 25, 2021 — The latest release of Sensu Go, version 6.6.0, is now available for download.

This release introduces PostgreSQL event store sorting as well as web UI improvements like support for ANSI color codes and a warning message when editing resources on a cluster with an older version than the gateway cluster. Sensu Go 6.6.0 also adds a label to logged metrics to help identify the backend that generated the metrics, logs connection errors along with context errors, and fixes a bug that could cause a backend crash in case of etcd client unavailability.

Read the upgrade guide to upgrade Sensu to version 6.6.0.

IMPROVEMENTS

- (Commercial feature) The web UI color themes are updated, and the default theme now uses cyan for elements like the left navigaton menu and breadcrumb navigation text.
- (Commercial feature) In the web UI, users now receive a warning message when they try to add or edit resources on an cluster that is running an older Sensu backend version than the

gateway cluster.

- (Commercial feature) The web UI now supports ANSI color codes, which improves check output readability when it includes color.
- (Commercial feature) Added support for sorting for the PostgreSQL event store. In addition, GraphQL can now use the PostgreSQL event store to sort events and get the total event count.
- Logged metrics now include a backend label. This makes it possible to associate metrics from the metrics log file with the backend they were generated on.
- Sensu no longer applies zero values for etcd configuration options. This prevents overwriting the etcd-provided default values with null, zero, slice, or empty values.

FIXES

- When sensu-go cannot connect to etcd, the connection error is now logged along with context errors.
- Fixed a bug that could cause sensu-backend to crash if the BackendIDGetter encounters etcd client unavailability.

6.5.5 release notes

November 22, 2021 — The latest release of Sensu Go, version 6.5.5, is now available for download.

The Sensu Go 6.5.5 patch release adds two backend configuration options for configuring the API and web UI HTTP servers' write timeouts and three new GraphQL duration metrics for the metrics log. This release also delivers several bug fixes, including fixes for sensu-backend and sensu-agent panics and failed keepalive lease grant operations.

Read the upgrade guide to upgrade Sensu to version 6.5.5.

IMPROVEMENTS

- Added the api-write-timeout and dashboard-write-timeout backend configuration options. These options allow you to configure the timeout for the respective HTTP servers' response writes, which is helpful when requests might take more than a few seconds to complete.
- Added graphql_duration_seconds, graphql_duration_seconds_sum, and graphql_duration_seconds_count to the metrics log. Also added objectives (0.5, 0.9, 0.99) to the graphql_duration_seconds metric.
- Added Prometheus metrics for tracking lease operations, with labels for operation type and

status, and added sensu_go_lease_ops to the metrics log.

FIXES

- Updated the assets, pipeline, and eventd duration metrics added in Sensu Go 6.5.2 to use milliseconds for consistency with other duration metrics.
- Updated the /version API so that responses reflect the versions of external etcd clusters based on the first available etcd endpoint.
- Fixed a bug that could cause sensu-backend and sensu-agent to panic due to concurrent websocket writes.
- Sensu no longer logs an error when one side of a websocket tries to close a previously closed connection.
- Sensu now retries keepalive lease grant operations that fail due to rate limiting.

6.5.4 release notes

October 30, 2021 — The latest release of Sensu Go, version 6.5.4, is now available for download.

This patch releases and updates the sensu-go core/api module.

Read the upgrade guide to upgrade Sensu to version 6.5.4.

FIXES

- Released and updated the sensu-go core/api module.

6.5.3 release notes

October 29, 2021 — The latest release of Sensu Go, version 6.5.3, is now available for download.

This patch adds the 6.5.2 metrics to the metrics log, fixes bugs in validation for environment variables in JavaScript mutators and asset expansion error handling, and vendors the correct version of sensu-go.

Read the upgrade guide to upgrade Sensu to version 6.5.3.

IMPROVEMENTS

- Added the Sensu Go 6.5.2 eventd, pipeline, and asset metrics to the metrics log to facilitate troubleshooting.

FIXES

- (Commercial feature) Vendored the correct version of sensu-go.
- Fixed a bug in API validation that rejected JavaScript mutators that use environment variables available in the environment rather than defined in the mutator env_vars attribute.
- Fixed a bug that prevented asset expansion errors from being handled.

6.5.2 release notes

October 28, 2021 — The latest release of Sensu Go, version 6.5.2, is now available for download.

The Sensu Go 6.5.2 patch release adds a number of metrics to provide more information about event handling and asset fetching and extraction. In addition, this patch makes operating system environment variables accessible with JavaScript mutators and omits an extraneous attribute that appeared in web UI resource data.

Read the upgrade guide to upgrade Sensu to version 6.5.2.

IMPROVEMENTS

- For JavaScript mutators, you can now list the names of any environment variables that are available in your environment (in addition to defining environment variables) in the env_vars attribute. This allows you to transform events with metatdata from the Sensu environment, which is useful for downstream processing and filtering when sending Sensu event data for further processing.
- Added sensu_go_event_handler_duration_sum and sensu_go_event_handler_duration_count and added `status` and `event_type` labels to the sensu_go_event_handler_duration metric. These updates allow you to determine whether an event was handled successfully.
- Added summary metrics for pipelined and pipeline operations to provide insight into how time is spent when pipelined and a pipeline resource handles an event:
    - sensu_go_pipelined_message_handler_duration
    - sensu_go_pipeline_duration
    - sensu_go_pipeline_resolve_duration

        - sensu_go_pipeline_filter_duration
        - sensu_go_pipeline_mutator_duration
        - sensu_go_pipeline_handler_duration
- Added summary metrics for eventd to provide insight into how time is spent when eventd handles an event:
    - sensu_go_eventd_create_proxy_entity_duration
    - sensu_go_eventd_update_event_duration
    - sensu_go_eventd_bus_publish_duration
    - sensu_go_eventd_liveness_factory_duration
    - sensu_go_eventd_switches_alive_duration
    - sensu_go_eventd_switches_bury_duration
- Added two metrics to provide insight into how time is spent when an asset is fetched and extracted:
    - sensu_go_asset_fetch_duration
    - sensu_go_asset_expand_duration

FIXES

- (Commercial feature) Fixes a bug in the web UI that added a `__virtual` attribute in the resource data for configuration resources.

6.5.1 release notes

October 20, 2021 — The latest release of Sensu Go, version 6.5.1, is now available for download.

This patch fixes several issues in the web UI and adds Prometheus counters for pipeline workflow handler processing.

Read the upgrade guide to upgrade Sensu to version 6.5.1.

IMPROVEMENTS

- Added Prometheus counters for pipeline workflow handler processing:
    - sensu_go_handler_requests: Number of processed handler requests

- sensu_go_handler_requests_total: Total number of handler requests invoked

FIXES

- (Commercial feature) Fixed a bug that could result in an error when listing entities with the PostgreSQL store enabled.
- (Commercial feature) In the web UI, fixed an issue with the native date and time picker that could cause problems when creating silences.
- (Commercial feature) In the web UI, fixed a bug that prevented users from editing service components.
- (Commercial feature) In the web UI, fixed the redirect for deleting entities so that it returns users to the Entities page rather than loading a 404 page for the deleted entity's details.

6.5.0 release notes

October 13, 2021 — The latest release of Sensu Go, version 6.5.0, is now available for download.

This release includes a number of improvements, specifically exciting new capabilities in the observability pipeline and a major simplification to how "pipelines" are configured. Sensu Go 6.5.0 introduces a new first-class `Pipeline` resource for defining logical pipeline workflows composed of filters + mutators + handlers. We're also introducing new streaming handler types: a TCPStreamHandler with TLS support and a SumoLogicMetricsHandler for seamless integration with the Sumo Logic Continuous Intelligence platform. Enhancements in the web UI include a completely overhauled configuration management system (with new views for the Checks, Filters, Handlers, and Mutators pages) and behind-the-scenes improvements that pave the way for even more new configuration management capabilities in future releases. Read the full release notes below for all the details!

Read the upgrade guide to upgrade Sensu to version 6.5.0.

NEW FEATURES:

- (Commercial feature) Added Sensu Plus, a built-in integration you can use to transmit your Sensu observability data to Sumo Logic via the Sumo Logic HTTP Logs and Metrics Source.
- (Commercial feature) Added support for Sumo Logic metrics handlers and TCP stream handlers. The enterprise/pipeline/v1 API endpoints provide HTTP access for retrieving and configuring Sumo Logic metrics handlers and TCP stream handlers.
- (Commercial feature) You can now view resource data for events, entities, and configuration

resources like checks and handlers directly in the web UI.

- (Commercial feature) In the web UI, you can execute individual checks on demand either according to existing subscriptions or on specific agents by adding and removing subscriptions without making changes to the saved check subscriptions.
- (Commercial feature) Added Prometheus metrics for TCP stream handlers:
  - sensu_go_tcp_stream_handler_events: Total number of events handled by the TCP stream handler
  - sensu_go_tcp_stream_handler_errors: Total number of errors produced by the TCP stream handler
  - sensu_go_tcp_stream_handler_latency: Distribution of handler latencies, in milliseconds, for the TCP stream handler
  - sensu_go_tcp_stream_handler_connection_acquisition_latency: Distribution of connection acquisition latencies (how long it takes to acquire a connection from the connection pool), in milliseconds, within the TCP stream handler
- New pipelines resource allows you to specify event filters, mutators, and handlers in a single workflow instead of listing filters and mutators in handler definitions. You can reference pipelines in your check definitions. The `/pipelines` API endpoint provides HTTP access for retrieving pipeline data and configuring pipelines, and you can use sensuctl to manage pipelines. Upgrade your Sensu agents to Sensu Go 6.5.0 to use pipelines resources.
- JavaScript mutators are now available. JavaScript mutators are evaluated by the Otto JavaScript VM as JavaScript programs, which enables greater throughput at scale than pipe mutators.
- Check definitions now include the pipelines attribute for specifying pipeline resources to use for the check's observability events.
- Added platform metrics logging to log core Sensu metrics in InfluxDB Line Protocol format, along with the `disable-platform-metrics`, `platform-metrics-log-file`, and `platform-metrics-logging-interval` backend configuration options for managing the platform metrics logging feature.
- Event logging is no longer a commercial-only feature.
- You can now set sensuctl environment variables for a single sensuctl command or with sensuctl configure.

IMPROVEMENTS:

- Added environment variables `SENSU_BACKEND_ETCD_CLIENT_USERNAME` and `SENSU_BACKEND_ETCD_CLIENT_PASSWORD` for connecting to external etcd via username and password authentication instead of certificate authentication. There are no corresponding

command line flags — these configuration options must be set via environment variables.

- You can now add an API key when you initialize the backend to make automated cluster setup and deployment more straightforward.
- Events now include the name of the agent that processed the event in the `processed_by` attribute to help you determine which agent processed an event executed by a proxy check request or a POST request to the events API.
- Added the `ignore-already-initialized` backend configuration option, which you can use to suppress the “already initialized” response and return an exit code 0 if a cluster has already been initialized.
- Upgraded Go version from 1.16.5 to 1.17.1.

SECURITY:

- Migrated dgrijalva/jwt-go to golang-jwt/jwt to address a vulnerability that would allow attackers to bypass intended access restrictions in situations. Read CVE-2020-26160 for more information.

FIXES:

- Sensuctl env now properly lists `SENSU_API_KEY` and `SENSU_TIMEOUT` as options for exporting environment variables. In addition, sensuctl command exec now properly adds the `SENSU_API_KEY` and `SENSU_TIMEOUT` variables to the command’s environment.
- Fixed a bug that could cause a crash when running the backend on darwin/arm64 and compressing a wrapped resource.
- Fixed a bug that could result in an etcd error if the number of silences in a given transaction exceeded etcd’s default maximum number of operations per transaction.

6.4.3 release notes

September 1, 2021 — The latest release of Sensu Go, version 6.4.3, is now available for download.

This patch fixes a deadlock in the event log writer.

Read the upgrade guide to upgrade Sensu to version 6.4.3.

FIXES:

- (Commercial feature) Fixed a bug that caused a deadlock in the event log writer.

6.4.2 release notes

August 31, 2021 — The latest release of Sensu Go, version 6.4.2, is now available for download.

This patch adds a backend configuration attribute that allows parallel event log encoding, as well as two summary metrics for the /metrics API endpoint.

Read the upgrade guide to upgrade Sensu to version 6.4.2.

FIXES:

- (Commercial feature) Added the `event-log-parallel-encoders` backend configuration attribute, which allows you to indicate whether Sensu should use parallel JSON encoders for event logging instead of the default (a single JSON encoding worker). This fixes a bottleneck in the event logging feature.

IMPROVEMENTS:

- Added sensu_go_agentd_event_bytes and sensu_go_store_event_bytes summary metrics to the /metrics API endpoint. sensu_go_agentd_event_bytes tracks the sizes of events, in bytes, received by agentd on the backend. sensu_go_store_event_bytes tracks event sizes, in bytes, received by the etcd store on the backend.

6.4.1 release notes

August 25, 2021 — The latest release of Sensu Go, version 6.4.1, is now available for download.

This patch includes fixes that improve forward- and backward-compatibility for backends and prevent `sensuctl cluster member-list` crashes, as well as changes to the default log levels for webd, the API, and the sensu-agent.

Read the upgrade guide to upgrade Sensu to version 6.4.1.

FIXES:

- (Commercial feature) For LDAP configurations, the `allowed_groups` attribute is omitted if not populated. This change improves backend reliability with older versions of federation and sensuctl.

- Fixed a bug to prevent `sensuctl cluster member-list` crashes when the etcd response header is nil.
- Fixed a `sensu-backend init` regression that returned exit status 0 if the store was already initialized.
- Sensu Go OSS can now be built on darwin/arm64.

IMPROVEMENTS:

- (Commercial feature) The default webd log level is now `warn`.
- The default log level for the Sensu API and `sensu-agent` is now `warn` (instead of `info`).
- The sensu-backend now reports when it is ready to process events at the `warn` level.
- You can now create resources with fields that are unknown to Sensu. This change improves forward-compatibility with newer Sensu backends.

6.4.0 release notes

June 28, 2021 — The latest release of Sensu Go, version 6.4.0, is now available for download.

The latest release of Sensu Go, version 6.4.0, is now available for download. This release includes a number of feature improvements and important bug fixes. We upgraded the embedded etcd from version 3.3 to 3.5 for improved stability and security. The `sensu-backend init` command now supports a `wait` flag, which indicates that the backend should repeatedly try to establish a connection to etcd until it is successful – fantastic news for Kubernetes users who want to bootstrap new Sensu Go clusters with external etcd! Check timeout also now works properly on Windows hosts: the Sensu Go agent can terminate check sub-processes on check execution timeout. This release fixes a bug that prevented deregistration events from working. There's something for everyone in this release!

Read the upgrade guide to upgrade Sensu to version 6.4.0.

NEW FEATURES:

- (Commercial feature) In the web UI, the system information modal now includes license expiration information, accessed via the `CTRL .` keyboard shortcut, for users with the appropriate permissions.
- (Commercial feature) Added page-specific configuration options and a custom sign-in message attribute for the web UI.

- Added binary-only distribution for macOS arm64.

IMPROVEMENTS:

- Added etcd-log-level backend configuration option for setting the log level for the embedded etcd server.
- Added `wait` flag for the `sensu-backend init` command, which indicates the backend should repeatedly try to establish a connection to etcd until it is successful.
- The `timeout` flag for `sensu-backend init` is now treated as a duration instead of seconds (example duration format is `10s` for 10 seconds or `5m` for 5 minutes). Values less than 1 second and integer values will be interpreted as seconds.
- Added `sensu_go_keepalives` Prometheus metric to count keepalive statuses over time and help identify instability due to keepalive failure.
- Upgraded Go version from `1.13.15` to `1.16.5`.
- Upgraded etcd version from `3.3.22` to `3.5.0`. As a result, **6.4.0 is not backward-compatible with previous Sensu versions**. Read the upgrade instructions for details about creating a full etcd database backup before you upgrade to Sensu Go 6.4.0. Also, in etcd 3.5, some Prometheus metric names changed. Read the etcd documentation for details.

FIXES:

- (Commercial feature) Selector statements that begin with quotes no longer cause an error if they follow the `&&` operator.
- (Commercial feature) Fixed a bug that allowed PostgresConfig resources to include a namespace attribute. Also, invalid PostgresConfig resources can no longer be created.
- Fixed a bug that resulted in OK keepalive status after shutting down the agent.
- Fixed a bug in role-based access control (RBAC) that caused incorrect HTTP API statuses and web UI crashes when role bindings referred to missing roles. The API now returns status `403` with a message to explain that the referenced role is missing.
- Fixed a bug that prevented deregistration events from validating due to empty `event.check.subscriptions` arrays.
- Fixed a bug that caused Windows agents to handle command timeouts improperly.

6.3.0 release notes

May 26, 2021 — The latest release of Sensu Go, version 6.3.0, is now available for download.

This release includes several new features, enhancements, bug fixes, and usability improvements. Construct a top-level business service-centric view for distributed infrastructure and applications with a preview of Business Service Monitoring! Rate-limit Sensu Go agent transport connections without using a separate load balancer. Use an API key to authenticate sensuctl, which is handy when automating Sensu Go configuration (for example CI pipelines) and other actions (like ad hoc check execution requests). The 6.3.0 release also improves the PostgreSQL store batching capabilities, raising the event processing throughput ceiling for most deployments. Check out the release notes below for more details — there's so much to love about this release!

Read the upgrade guide to upgrade Sensu to version 6.3.0.

NEW FEATURES:

- (Commercial feature) Added business service monitoring (BSM) to provide high-level visibility into the current health of any number of business services, with a built-in aggregate check rule template.
- (Commercial feature) Added support for agent transport rate limiting via `agent-burst-limit` and `agent-rate-limit` backend configuration options.
- (Commercial feature) Added the `event-log-buffer-wait` backend configuration option, which allows you to specify how long the event logger will wait for the writer to consume events from the buffer when the buffer is full.
- Added the entity class service, which represents a business service for the business service monitoring (BSM) feature.

IMPROVEMENTS:

- (Commercial feature) The agent transport health API endpoint repsonse now includes PostgreSQL health information.
- (Commercial feature) Added the `poll_interval` default preferences attribute to the `GlobalConfig` resource so administrators can adjust how often the web UI pages poll for new data.
- (Commercial feature) In the web UI, some form fields now include examples of valid values.
- Added the `--api-key` global flag for sensuctl commands. Use this flag with sensuctl commands to bypass username/password authentication.
- Logs for JavaScript filter evaluation errors now include more context.
- Concatenated YAML files now support carriage return and line feed (CRLF).
- Removed extraneous shell auto-completion suggestions for sensuctl.

FIXES:

- (Commercial feature) Migrated the PostgreSQL event store from github.com/lib/pq to github.com/jackc/pgx so that PostgreSQL batching works properly.
- (Commercial feature) In the web UI, error messages are now visible in dark mode.
- Fixed a bug that could cause the scheduler to crash when using round robin checks.
- Fixed a bug that calculated build information for every keepalive in OSS builds.
- SIGHUP no longer triggers an internal restart.

6.2.7 release notes

April 1, 2021 — The latest release of Sensu Go, version 6.2.7, is now available for download.

This patch includes fixes for potential deadlocks in metricsd and agentd and crashes in the scheduler and tessend as well as for a bug that calculated build information for every keepalive.

Read the upgrade guide to upgrade Sensu to version 6.2.7.

FIXES:

- (Commercial feature) Fixed a potential deadlock in metricsd that could occur when performing an internal restart.
- Fixed a potential deadlock in agentd due to the unit test timing out in the build pipeline.
- Fixed a bug that could cause the scheduler to crash when using round robin checks.
- Fixed a bug that calculated build information for every keepalive in OSS builds.
- Fixed a potential crash in tessend that could occur if the `ringv2.Event.Value` has a zero length.
- Fixed a bug that allowed some etcd watchers to try to process watch events that contain invalid pointers.

6.2.6 release notes

March 25, 2021 — The latest release of Sensu Go, version 6.2.6, is now available for download.

This patch fixes a bug that allowed PostgreSQL round robin scheduling to use a separate PostgreSQL connection for each subscription and improves the validation for POST/PUT requests for enterprise API endpoints.

Read the upgrade guide to upgrade Sensu to version 6.2.6.

FIXES:

- (Commercial feature) Fixed a bug that allowed PostgreSQL round robin scheduling to use a separate PostgreSQL connection for each subscription. PostgreSQL round robin scheduling now uses exactly one extra PostgreSQL connection.
- (Commercial feature) Improved the validation for POST/PUT requests for enterprise API endpoints. Sensu now checks the type and namespace in the request body against the type and namespace in the request URL.

6.2.5 release notes

February 2, 2021 — The latest release of Sensu Go, version 6.2.5, is now available for download.

This patch fixes a bug regarding the event occurrences_watermark property. This bug interfered with the property's expected behavior when using event filters like the popular fatigue check filter.

Read the upgrade guide to upgrade Sensu to version 6.2.5.

FIXES:

- (Commercial feature) Fixed a bug that prevented occurrences_watermark from incrementing for non-zero events when using the PostgreSQL datastore.

6.2.4 release notes

January 28, 2021 — The latest release of Sensu Go, version 6.2.4, is now available for download.

This patch fixes a bug that prevented `federation/v1.Cluster` from appearing in the response for `sensuctl describe-type all` and resolves a web UI performance issue for PostgreSQL users.

Read the upgrade guide to upgrade Sensu to version 6.2.4.

FIXES:

- (Commercial feature) `federation/v1.Cluster` now appears in the `sensuctl describe-type all` response.
- (Commercial feature) Fixed a performance issue that affected the web UI when using the PostgreSQL datastore.

6.2.3 release notes

January 21, 2021 — The latest release of Sensu Go, version 6.2.3, is now available for download.

This patch fixes two bugs: one that could prevent the `agent-managed-entity` configuration option from working properly and one that caused `sensuctl dump` output to include events from all namepaces rather than the specified namespace.

Read the upgrade guide to upgrade Sensu to version 6.2.3.

FIXES:

- Fixed a bug that prevented the agent-managed-entity configuration attribute from working properly when no labels are defined.
- Fixed a bug where `sensuctl dump` output included events from all namespaces the user had access permissions for rather than events from only the specified namespace.

6.2.2 release notes

January 14, 2021 — The latest release of Sensu Go, version 6.2.2, is now available for download.

This patch fixes bugs that prevented PostgreSQL round robin scheduling from working properly.

Read the upgrade guide to upgrade Sensu to version 6.2.2.

FIXES:

- (Commercial feature) Fixed a bug that could improperly enable PostgreSQL round robin scheduling after creating a PostgreSQL configuration.
- (Commercial feature) Fixed a bug that prevented PostgreSQL round robin scheduling if the

namespace and check names were more than 63 characters long, combined.

6.2.1 release notes

January 11, 2021 — The latest release of Sensu Go, version 6.2.1, is now available for download.

This patch fixes bugs that could prevent users from enabling PostgreSQL after upgrading from 5.x or configuring agent labels and annotations with flags. In addition, `sensuctl prune hook` and `sensuctl prune check` now work as expected and users can no longer edit agent-managed entities in the web UI.

Read the upgrade guide to upgrade Sensu to version 6.2.1.

FIXES:

- (Commercial feature) Fixed a bug that prevented users from enabling PostgreSQL as the event store after upgrading from 5.x.
- (Commercial feature) The `sensuctl prune hook` and `sensuctl prune check` subcommands now work as expected.
- (Commercial feature) In the web UI, fixed a bug that allowed users to edit Sensu agent-managed entities.
- Fixed a bug that generated a small amount of extra etcd or PostgreSQL traffic upon keepalive failure.
- In silenced entries, the `expire` field now represents the configured number of seconds until the entry should be deleted rather than the entry's remaining duration.
- Labels and annotations are now configuration options for sensu-agent.

6.2.0 release notes

December 17, 2020 — The latest release of Sensu Go, version 6.2.0, is now available for download.

The latest release of Sensu Go, version 6.2.0, is now available for download! Sensu Go 5.x and configuration management users rejoice: this release adds support for agent local configuration (that is, agent.yml) managed entities! Agent entities may now be managed exclusively by their agents when `sensu-agent` is started with the new `agent-managed-entity` configuration option. This makes it more straightforward to migrate from Sensu Go 5.x to 6.x, as existing agent entity management workflows like Puppet will just work with the new option enabled! Note that you will not be able to edit

agent-managed entities via the backend REST API or web UI.

Sensu Go 6.2.0 includes significant feature enhancements such as PostgreSQL backend round robin check scheduling for increased reliability and consistency, an updated format for silenced entry dates and durations in sensuctl tabular-format output, and a /health API endpoint for agent WebSocket transport status. This release delivers important bug fixes like consistently using `event_id` in logs and eliminating the sensuctl error when Vault provider SSL certificates do not exist on the local system. Also, enterprise/prune/v1alpha no longer requires cluster-wide permissions; users with limited permissions can put it to use in their namespaces!

Read the upgrade guide to upgrade Sensu to version 6.2.0.

NEW FEATURES:

- (Commercial feature) Added support for the `memberof` attribute for the LDAP authentication provider.
- (Commercial feature) Added the ability to exclude resource types when using sensuctl prune with the –omit flag.
- (Commercial feature) Added support for round robin scheduling on PostgreSQL instead of etcd.
- (Commercial feature) Added support for OIDC authentication via sensuctl configure.
- Entities may now be managed exclusively by their agents when sensu-agent is started with the agent-managed-entity configuration attribute.
- The /metrics API endpoint now exposes build information as a Prometheus metric.
- Added /health API endpoint to agent WebSocket transport.
- Checks now include the `scheduler` attribute, which Sensu automatically populates with the type of scheduler that schedules the check.
- Events now include the `sequence` attribute, which the Sensu agent automatically sets at startup and increments by 1 at every successive check execution or keepalive event.
- Added support for using environment variables to define the configuration file paths for the Sensu agent (`SENSU_CONFIG_FILE`) and backend (`SENSU_BACKEND_CONFIG_FILE`).

IMPROVEMENTS:

- (Commercial feature) Refactored entity limiter to ensure that warning messages about approaching a license's entity or entity class limit are now only displayed for users with `create` or `update` permissions for the license.
- (Commercial feature) The [enterprise/prune/v1alpha API] endpoints194 and the sensuctl

interface now require less-broad permissions.

- Adjusted the format for silenced entry dates and durations in sensuctl tabular-format output. For all silenced entries, the begin date is now listed in RFC 3339 format. For silenced entries that have not begun, the list displays the expiration date in RFC 3339 format. For silenced entires with no expiration date, the list displays `-1`. For silenced entries that have begun, the list displays the duration (for example, 1m30s).
- Sensuctl and sensu-backend now ask users to retype their passwords when creating a new password in interactive mode.

FIXES:

- (Commercial feature) Sensuctl no longer produces an error when SSL certificates for the Vault provider do not exist on the local system.
- Logs now consistently use `event_id` rather than `event_uuid`.
- Sensuctl commands that only contain subcommands now exit with status code 46 when no arguments or incorrect arguments are given.
- The sensuctl dump command now includes a description.
- Sensuctl command descriptions now have consistent capitalization.
- Use of the `config-file` flag is no longer order-dependent.

6.1.4 release notes

December 16, 2020 — The latest release of Sensu Go, version 6.1.4, is now available for download.

This patch fixes a bug that could cause a crash in the backend API, addresses a case where agents do not honor HTTP proxy environment variables, and improves the error message reported by the agent when asset checksums do not match expectations.

Read the upgrade guide to upgrade Sensu to version 6.1.4.

FIXES:

- Fixed a bug that could cause a panic in the backend core/v2/entities API.
- The agent asset fetching mechanism now respects HTTP proxy environment variables when `trusted-ca-file` is configured.
- When an asset artifact retrieved by the agent does not match the expected checksum, the

logged error now includes the size of the retrieved artifact and more clearly identifies the expected and actual checksums.

6.1.3 release notes

November 9, 2020 — The latest release of Sensu Go, version 6.1.3, is now available for download.

This patch fixes a bug that caused event updates to fail with an error about a null value in the occurrences column. This bug only affects Sensu instances that use PostgreSQL as the event store.

Read the upgrade guide to upgrade Sensu to version 6.1.3.

FIXES:

- (Commercial feature) For instances that use PostgreSQL as the event store, fixed a bug that caused event updates to fail with an error message about a null value in the occurrences column.

6.1.2 release notes

October 28, 2020 — The latest release of Sensu Go, version 6.1.2, is now available for download.

This patch release resolves a backend and agent crash related to JavaScript execution.

Read the upgrade guide to upgrade Sensu to version 6.1.2.

FIXES:

- Fixed a bug related to JavaScript execution that could cause a crash in the backend and agent.

6.1.1 release notes

October 22, 2020 — The latest release of Sensu Go, version 6.1.1, is now available for download.

This patch release includes a number of bug fixes that affect proper hook handling with `sensuctl prune` and `sensuctl dump`, entity creation via `sensuctl create`, form validation for subscription names in the web UI, and permissions for PATCH requests, among other fixes.

Read the upgrade guide to upgrade Sensu to version 6.1.1.

FIXES:

- (Commercial feature) `sensuctl prune` now properly handles hooks when pruning resources.
- (Commercial feature) Fixed a bug that returned incorrect `!=` results for label selectors when no labels were defined.
- (Commercial feature) In the web UI, fixed a bug that could cause a GraphQL `no claims` error when a user's access token was no longer valid instead of displaying the sign-out dialog window.
- (Commercial feature) In the web UI, form validation for subscription names now matches allowed values.
- Fixed a bug that prevented sensu-agent from shutting down correctly.
- Entities are now properly created using `sensuctl create`.
- Per-entity subscriptions now persist with PATCH requests.
- Any user with `update` permissions for a resource can now make PATCH requests for that resource.
- HookConfig can now be exported via `sensuctl dump`. Also, `sensuctl dump` now properly logs API errors.
- eventd errors now include additional context for debugging.

6.1.0 release notes

October 5, 2020 — The latest release of Sensu Go, version 6.1.0, is now available for download.

This release delivers significant performance and stability gains, feature enhancements, and several bug fixes. The web UI is now much snappier, and its search is redesigned with an improved syntax and suggestions! Monitor even more services and infrastructure when using the PostgreSQL store: batched Sensu event writes and improved indexing allows a single Sensu Go deployment to process and query more data than ever before. If you're using Prometheus client libraries to instrument your applications, the Sensu Go agent can now scrape and enrich those metrics! And if you're collecting metrics in other formats like Nagios PerfData, you can use the new output metric tags feature to enrich those metrics too! The sensuctl prune command also received some love, and it now loads and prunes configuration resources from multiple files!

Read the upgrade guide to upgrade Sensu to version 6.1.0.

NEW FEATURES:

- (Commercial feature) Added support for custom secrets engine paths in Vault secrets.
- (Commercial feature) In the web UI, added new search functionality, with improved syntax and suggestions.
- (Commercial feature) Added `strict` attribute to the PostgresConfig type to help debug incorrect configurations and database permissions.
- (Commercial feature) Added `batch_buffer`, `batch_size`, and `batch_workers` attributes to the PostgresConfig type so operators can optimize PostgreSQL latency and throughput.
- (Commercial feature) Added TLS configuration to the cluster resource so you can specify additional CA certificates and insecure mode.
- (Commercial feature) Added a `types` query parameter for listing authentication providers and secrets providers via the API.
- (Commercial feature) Added the Sensu SaltStack Enterprise Handler for launchingSaltStack Enterprise Jobs for automated remediation.
- (Commercial feature) The Alpine-based Docker image now has multi-architecture support with support for the linux/386, linux/amd64, linux/arm64, linux/arm/v6, linux/arm/v7, linux/ppc64le, and linux/s390x platforms.
- The backend configuration option `api-request-limit` is now available to configure the maximum API request body size (in bytes).
- In the REST API, most configuration resources now support the PATCH method for making updates.
- Added new handler and check plugins: Sensu Go Elasticsearch Handler, Sensu Rundeck Handler, and Sensu Kubernetes Events Check.

IMPROVEMENTS:

- (Commercial feature) Improved logging for OIDC authentication providers. Also added `disable_offline_access` OIDC spec attribute, which provides a workaround for authorization servers that do not support the `offline_access` scope.
- (Commercial feature) Added indexed field and label selectors to the PostgreSQL event store to improve performance for PostgreSQL event store queries with field and label selectors.
- Added Prometheus transformer for extracting metrics from check output using the Prometheus Exposition Text Format.
- Added the `output_metric_tags` attribute for checks so you can apply custom tags to enrich

metric points produced by check output metric extraction.

- A warning is now logged when you request a dynamic runtime asset that does not exist.
- The trusted CA file is now used for agent, backend, and sensuctl asset retrieval.
- Per-entity subscriptions (such as `entity:entityName`) are always available for agent entities, even you remove subscriptions via the core/v2/entities API endpoints.
- Updated the Sensu TimescaleDB Handler to write tags as a JSON object instead of an array of objects, which facilitates tags queries.
- Updated the Sensu Go Data Source for Grafana plugin to support using API keys, fetching resources from all namespaces, using Sensu's built-in resposne filtering, grouping aggregation results by attribute, and number of other improvements.

FIXES:

- (Commercial feature) Fixed a bug in `sensuctl dump` that allowed polymorphic resources (e.g., secrets providers and authentication providers) to dump other providers of the same type.
- (Commercial feature) Check output is no longer truncated in the event log file when the max output size is set and the PostgreSQL event store is enabled.
- (Commercial feature) Sensuctl prune now handles multi-file/multi-url input correctly.
- (Commercial feature) Fixed a bug where PostgreSQL errors could cause the backend to panic.
- (Commercial feature) Fixed a bug where PostgreSQL would refuse to store event with a negative check status.
- The backend will no longer start if the web UI TLS configuration is not fully specified.
- The agent entity is now included in data passed to the stdin for the command process.
- Improved check scheduling to prevent stale proxy entity data when using cron or round robin schedulers.
- Fixed a bug that resulted in incorrect entity listings for agent entities created via the API instead of sensu-agent.
- When downloading assets, Sensu now closes the response body after reading from it.
- Fixed a crash in the backend and agent related to JavaScript execution.

6.0.0 release notes

August 10, 2020 — The latest release of Sensu Go, version 6.0.0, is now available for download.

With Sensu Go 6.0.0, you can control everything through the API. You can still use configuration management tools to bootstrap agent entities, but you don't need to! Our new agent entity management feature via the backend configuration API nearly eliminates the need for external (or out-of-band) configuration management for Sensu, which allows you to manage agent entity subscriptions and automate the discovery of system facts without updating agent local configuration files. Run a sensuctl command, click a button in the web UI, or execute a custom check plugin!

Read the upgrade guide to upgrade Sensu to version 6.0.0.

BREAKING CHANGES FOR SENSU 6.0:

- The database schema for entities has changed. As a result, after you complete the steps to upgrade to Sensu 6.0 (including running the `sensu-backend upgrade` command), you will not be able to use your database with older versions of Sensu.
- For Sensu Go instances built from source, the web UI is now a standalone product — it is no longer included with the Sensu backend. Visit the Sensu Go Web repository for more information.
- After initial creation, you cannot change your `sensu-agent` entity configuration by modifying the agent's configuration file.

NEW FEATURES:

- (Commercial feature) Sensu now logs a warning when secrets cannot be sent to an agent because mTLS is not enabled.
- (Commercial feature) Added JavaScript functions `sensu.EventStatus`, `sensu.FetchEvent`, and `sensu.ListEvents` to the filter execution environment so you can now query the Sensu event store for other events within the filter namespace.
- (Commercial feature) Docker-only Sensu now binds to the hostname of containers instead of `localhost`. Docker images now set their own default values for environment variables `SENSU_AGENT_API_URL`, `SENSU_BACKEND_API_URL`, `SENSU_BACKEND_ETCD_INITIAL_CLUSTER`, `SENSU_BACKEND_ETCD_ADVERTISE_CLUSTER`, `SENSU_BACKEND_ETCD_INITIAL_ADVERTISE_PEER_URLS`, `SENSU_BACKEND_ETCD_LISTEN_CLIENT_URLS`, and `ETCD_LISTEN_PEER_URLS`.
- (Commercial feature) Added Linux packages for 386; armv5, armv6, and armv7; MIPS hard float, MIPS LE hard float, and MIPS 64 LE hard float; ppc64le; and s390x architectures. Review the supported platforms page for a complete list of Sensu's supported platforms.
- (Commercial feature) Added Sensu query expression `sensu.CheckDependencies`.
- Added binary-only distributions for FreeBSD `armv5`, `armv6`, and `armv7` and Linux `ppc64le` and `s390x`.

- Added the `is_silenced` Boolean attribute to the event.Check object to indicate whether the event was silenced at the time it was processed.

IMPROVEMENTS:

- (Commercial feature) Added support for the `memberOf` attribute in Active Directory (AD).
- (Commercial feature) Added more descriptive information for errors in the federated web UI.
- The `dead` and `handleUpdate` methods in keepalived now use `EntityConfig` and `EntityState` respectively.
- The `dead()` and `createProxyEntity()` methods in eventd now use `corev3.EntityConfig` and `corev3.EntityState`.
- Agent entity updates now ignore state-related fields.
- You can now manage Sensu agent configuration via the HTTP API.
- For sysvinit services, Sensu now passes users' secondary groups (that is, groups other than the Sensu user group) to `chroot`, which gives the Sensu agent and backend access to the file access writes that are granted to the secondary groups.
- Output of `sensuctl asset add` now includes help for using the runtime asset.
- For role bindings and cluster role bindings, `subjects.name` values can now include unicode characters, and `roleRef.type` and `subjects.type` values are now automatically capitalized.
- Improved logging for the agent WebSocket connection.
- Improved the wording of the secret provider error message.
- Fewer keys in etcd are now stored for agents.
- Keepalive and round robin scheduling leases are now dealt with more efficiently.
- Upgraded Go version from 1.13.7 to 1.13.15.
- Upgraded etcd version from 3.3.17 to 3.3.22.

FIXES:

- (Commercial feature) Label selectors now work as expected with multiple requirements for events.
- (Commercial feature) Fixed an issue that prevented broken secrets providers from surfacing their errors.
- (Commercial feature) Fixed a bug for PostgreSQL datastores that could prevent GraphQL from retrieving all pages when fetching events in a namespace with more than 1000 total events,

resulting in an unexpected error.

- (Commercial feature) Fixed a bug that could cause the backend to panic in case of PostgreSQL errors.
- Sensu now logs and returns and error if it cannot find a mutator.
- Errors produced in the agent by assets, check validation, token substitution, and event unmarshaling are logged once again.
- The User-Agent header is now set only upon on new client creation rather than upon each request.
- When the Sensu agent cannot parse the proper CA certificate path, Sensu logs this in the error message.
- Fixed a bug where highly concurrent event filtering could result in a panic.
- Fixed a bug where nil labels or annotations in an event filtering context would require you to explicitly check whether the annotations or labels are undefined. With this fix, labels and annotations are always defined (although they may be empty).
- Fixed the log entry field for the check's name in schedulerd.

5.21.5 release notes

March 25, 2021 — The latest release of Sensu Go, version 5.21.5, is now available for download.

The Sensu 5.21.5 patch release improves the validation for POST/PUT requests for enterprise API endpoints.

Read the upgrade guide to upgrade Sensu to version 5.21.5.

FIXES:

- (Commercial feature) Improved the validation for POST/PUT requests for enterprise API endpoints. Sensu now checks the type and namespace in the request body against the type and namespace in the request URL.

5.21.4 release notes

March 9, 2021 — The latest release of Sensu Go, version 5.21.4, is now available for download.

This patch release fixes a bug that caused the SIGHUP signal to restart the sensu-backend.

Read the upgrade guide to upgrade Sensu to version 5.21.4.

FIXES:

- Fixed a bug that caused the SIGHUP signal used for log rotation to restart the sensu-backend.

5.21.3 release notes

October 14, 2020 — The latest release of Sensu Go 5, version 5.21.3, is now available for download.

This patch release includes a few fixes to improve stability and correctness.

Read the upgrade guide to upgrade Sensu to version 5.21.3.

FIXES:

- Fixed a bug where HTTP connections could be left open after downloading assets.
- Fixed a bug where event filter or asset filter execution could cause a crash.
- (Commercial feature) Fixed a bug where PostgreSQL would refuse to store event with a negative check status.

5.21.2 release notes

August 31, 2020 — The latest release of Sensu Go, version 5.21.2, is now available for download.

This patch release includes two fixes: one for PostgreSQL errors that could cause the backend to panic and one to ensure that failed check events are written to the event log file.

Read the upgrade guide to upgrade Sensu to version 5.21.2.

FIXES:

- (Commercial feature) Fixed a bug where PostgreSQL errors could cause the backend to panic.
- Failed check events are now written to the event log file.

5.21.1 release notes

August 5, 2020 — The latest release of Sensu Go, version 5.21.1, is now available for download.

This patch release includes fixes for a web UI crash when interacting with namespaces that contain 1000 or more events and regressions in logging various agent errors as well as an enhancement that provides additional context to WebSocket connection errors logged by the backend.

Read the upgrade guide to upgrade Sensu to version 5.21.1.

IMPROVEMENTS:

- Backend log messages related to connection errors on the agent WebSocket API now provide more context about the error.

FIXES:

- Fixed a potential web UI crash when fetching events in namespace with 1000 or more events.
- Fixed a regression that prevented errors produced in the agent by assets, check validation, token substitution, or event unmarshaling from being logged.

5.21.0 release notes

June 15, 2020 — The latest release of Sensu Go, version 5.21.0, is now available for download.

The latest release of Sensu Go, version 5.21.0, is now available for download! This release delivers several enhancements and fixes. The most significant enhancements involve user management: you can now generate a password hash, specify the resulting hash in your user definitions without having to store cleartext passwords, and create and update these users using `sensuctl create`. You can also reset user passwords via the backend API. We also tuned Sensu Go agent logging and changed the default log level from warning to info. Plus, we crushed a number of nasty bugs: checks configured with missing hooks can no longer crash the agent, proxy check request errors do not block scheduling for other entities, and more!

Read the upgrade guide to upgrade Sensu to version 5.21.0.

NEW FEATURES:

-

- (Commercial feature) Added entity count and limit for each entity class in the tabular title in the response for `sensuctl license info` (in addition to the total entity count and limit).
- (Commercial feature) Added Linux amd64 OpenSSL-linked binaries for the Sensu agent and backend, with accompanying `require-fips` and `require-openssl` configuration options for the agent and backend.
- Added `sensuctl user hash-password` command to generate password hashes.
- Added the ability to reset passwords via the backend API and `sensuctl user reset-password`.

IMPROVEMENTS:

- Changed the default log level for `sensu-agent` to `info` (instead of `warn`).

FIXES:

- The password verification logic when running `sensuctl user change-password` is now included in the backend API rather than sensuctl.
- Errors in publishing proxy check requests no longer block scheduling for other entities.
- Using the `--chunk-size` flag when listing namespaces in sensuctl now works properly.
- The agent no longer immediately exits in certain scenarios when components are disabled.
- Fixed a bug that could cause a GraphQL query to fail when querying a namespace that contained event data in excess of 2 GB.

5.20.2 release notes

May 26, 2020 — The latest release of Sensu Go, version 5.20.2, is now available for download.

This patch release adds username to the API request log to help operators with troubleshooting and user activity reporting, as well as validation for subjects in role-based access control (RBAC) role binding and cluster role binding.Release 5.20.2 also temporarily disables process discovery so we can investigate and resolve its performance impact on the backend (increased CPU and memory usage).

Read the upgrade guide to upgrade Sensu to version 5.20.2.

NEW FEATURES:

- The API request log now includes the username.

FIXES:

- (Commercial feature) Process discovery in the agent is temporarily disabled.
- The system's libc_type attribute is now properly populated for Ubuntu entities.
- Single-letter subscriptions are now allowed.
- Subjects are now validated in RBAC role binding and cluster role binding.
- Sensuctl command assets can now be retrieved and installed from Bonsai.

5.20.1 release notes

May 15, 2020 — The latest release of Sensu Go, version 5.20.1, is now available for download.

This patch release includes a bug fix that affects the web UI federated homepage gauges when using the PostgreSQL datastore and several fixes for the data displayed in the web UI entity details.

Read the upgrade guide to upgrade Sensu to version 5.20.1.

FIXES:

- (Commercial feature) Fixes a bug that prevented the federated homepage in the web UI from retrieving the keepalive and event gauges when PostgreSQL was configured as the event datastore.
- (Commercial feature) The memory_percent and cpu_percent processes attributes are now properly displayed in the web UI.
- In the web UI, the entity details page no longer displays float type (which applies only for MIPS architectures). Also on entity details pages, the system's libc type is now listed and process names are no longer capitalized.

5.20.0 release notes

May 12, 2020 — The latest release of Sensu Go, version 5.20.0, is now available for download.

This release delivers several new features, substantial improvements, and important fixes. One exciting new feature is agent local process discovery to further enrich entities and their events with valuable context. Other additions include a web UI federation view that provides a single pane of glass

for all of your Sensu Go clusters and token substitution for assets. And Windows users rejoice! This release includes many Windows agent fixes, as well as agent log rotation capabilities!

Read the upgrade guide to upgrade Sensu to version 5.20.0.

NEW FEATURES:

- (Commercial feature) Added a `processes` field to the system type to store agent local processes for entities and events and a `discover-processes` option to the agent configuration options to populate the `processes` field in entity.system if enabled.
- (Commercial feature) Added a new resource, `GlobalConfig`, that you can use to customize your web UI configuration.
- (Commercial feature) Added metricsd to collect metrics for the web UI and the `metrics-refresh-interval` backend configuration option for setting the interval at which Sensu should refresh metrics.
- (Commercial feature) Added process and additional system information to the entity details view in the web UI.
- (Commercial feature) Added a PostgreSQL metrics suite so metricsd can collect metrics about events stored in PostgreSQL.
- (Commercial feature) Added entity class limits to the license.
- Added check hook output to event details page in the web UI.
- Added the sensuctl describe-type command to list all resource types. The `sensuctl describe-type` command deprecates `sensuctl dump --types`.
- Added `annotations` and `labels` as backend configuration options.
- Added token substitution for assets.
- Added `event.is_silenced` and `event.check.is_silenced` field selectors.
- Added `Edition` and `GoVersion` fields to version information. In commercial distributions, the `Edition` version attribute is set to `enterprise`
- Added ability to configure the Resty HTTP timeout. Also, sensuctl now suppresses messages from the Resty library.

IMPROVEMENTS:

- (Commercial feature) The web UI homepage is now a federated view.
- You can now increment the log level by sending SIGUSR1 to the sensu-backend or sensu-agent process.

- License metadata now includes the current entity count and license entity limit.
- In the web UI, users will receive a notification when they try to delete an event without appropriate authorization.
- The Windows agent now has log rotation capabilities.
- Notepad is now the default editor on Windows rather than vi.

FIXES:

- (Commercial feature) Database connections no longer leak after queries to the cluster /health API.
- In the web UI, any leading and trailing whitespace is now trimmed from the username when authenticating.
- The web UI preferences dialog now displays only the first five groups a user belongs to, which makes the sign-out button more accessible.
- In the web UI, the deregistration handler no longer appears as `undefined` on the entity details page.
- You can now escape quotes to express quoted strings in token substitution templates.
- The Windows agent now accepts and remembers arguments passed to `service run` and `service install`.
- The Windows agent now synchronizes writes to its log file, so the file size will update with every log line written.
- The Windows agent now logs to both console and log file when you use `service run`.

5.19.3 release notes

May 4, 2020 — The latest release of Sensu Go, version 5.19.3, is now available for download. This is a patch release with many improvements and bug fixes, including a fix to close the event store when the backend restarts, a global rate limit for fetching assets, and fixes for goroutine leaks. Sensu Go 5.19.3 also includes several web UI updates, from fixes to prevent crashes to new color-blindness modes.

Read the upgrade guide to upgrade Sensu to version 5.19.3.

FIXES:

- The event store now closes when the backend restarts, which fixes a bug that allowed Postgres connections to linger after the backend restarted interally.

- The etcd event store now returns exact matches when retrieving events by entity (rather than prefixed matches).
- `sensu-backend init` now logs any TLS failures encountered during initialization.
- `sensuctl logout` now resets the TLS configuration.
- `env_vars` values can now include the equal sign.
- Error logs now include underlying errors encountered when fetching an asset.
- The log level is now WARNING when an asset is not installed because none of the filters match.
- Fixes a bug in the web UI that could cause labels with links to result in a crash.
- Fixes a bug in the web UI that could cause the web UI to crash when using an unregistered theme.
- Fixes a bug that could cause the backend to crash.
- Fixes a bug in multi-line metric extraction that appeared in Windows agents.
- Fixes an authentication bug that restarted the sensu-backend when agents disconnected.
- Fixes a bug that meant check `state` and `last_ok` were not computed until the second instance of the event.
- Fixes a bug that caused messages like “unary invoker failed” to appear in the logs.
- Fixes several goroutine leaks.
- Fixes a bug that caused the backend to crash when the etcd client received the error “etcdserver: too many requests.”

IMPROVEMENTS:

- In the web UI, color-blindness modes are now available.
- In the web UI, labels and annotations with links to images will now be displayed inline.
- Adds a global rate limit for fetching assets to prevent abusive asset retries, which you can configure with the `assets-burst-limit` and `assets-rate-limit` configuration options for the agent and backend.
- Adds support for restarting the backend via SIGHUP.
- Adds a timeout flag to `sensu-backend init`.
- Deprecated flags for `sensuctl silenced update` subcommand have been removed.

5.19.2 release notes

April 27, 2020 — The latest release of Sensu Go, version 5.19.2, is now available for download. This patch release adds two database connection pool parameters for PostgreSQL so you can configure the maximum time a connection can persist before being destroyed and the maximum number of idle connections to retain. The release also includes packages for Ubuntu 19.10 and 20.04.

Read the upgrade guide to upgrade Sensu to version 5.19.2.

FIXES:

- (Commercial feature) Adds SQL database connection pool parameters `max_conn_lifetime` and `max_idle_conns` to store/v1.PostgresConfig132.

IMPROVEMENTS:

- Sensu packages are now available for Ubuntu 19.10 (Eoan Ermine) and 20.04 (Focal Fossa). Review the supported platforms page for a complete list of Sensu's supported platforms and the installation guide to install Sensu packages for Ubuntu.

5.19.1 release notes

April 13, 2020 — The latest release of Sensu Go, version 5.19.1, is now available for download. This is a patch release with a number of bug fixes, including several that affect keepalive events, as well as an addition to the help response for `sensu-backend start` and `sensu-agent start`: the default path for the configuration file.

Read the upgrade guide to upgrade Sensu to version 5.19.1.

FIXES:

- (Commercial feature) Fixed a bug that caused the PostgreSQL store to be enabled too late upon startup, which caused keepalive bugs and possibly other undiscovered bugs.
- Keepalives now fire correctly when using the PostgreSQL event store.
- Keepalives can now be published via the HTTP API.
- `sensu-agent` no longer allows configuring keepalive timeouts that are shorter than the keepalive interval.
- Eventd no longer mistakes keepalive events for checks with TTL.

- Keepalives now generate a new event universally unique identifier (UUID) for each keepalive failure event.
- Agents now correctly reset keepalive switches on reconnect, which fixes a bug that allowed older keepalive timeout settings to persist.
- Token substitution templates can now express escape-quoted strings.
- The REST API now uses a default timeout of 3 seconds when querying etcd health.
- Pipe handlers now must include a command.
- The response for `sensu-backend start --help` and `sensu-agent start --help` now includes the configuration file default path.
- The system's `libc_type` attribute is now populated on Alpine containers.

5.19.0 release notes

March 30, 2020 — The latest release of Sensu Go, version 5.19.0, is now available for download. This release is packed with new features, improvements, and fixes, including our first alpha feature: declarative configuration pruning to help keep your Sensu instance in sync with Infrastructure as Code workflows. Other exciting additions include the ability to save and share your filtered searches in the web UI, plus a new `matches` substring match operator that you can use to refine your filtering results! Improvements include a new `created_by` field in resource metadata and a `float_type` field that stores whether your system uses hard float or soft float. We've also added agent and sensuctl builds for MIPS architectures, moved Bonsai logs to the `debug` level, and added PostgreSQL health information to the /health API payload.

Read the upgrade guide to upgrade Sensu to version 5.19.0.

NEW FEATURES:

- (Commercial feature) In the web UI, you can now save, recall, and delete filtered searches.
- (Commercial feature) Added the `matches` substring matching operator for API response, sensuctl, and web UI filtering selectors.
- (Commercial feature) Added agent and sensuctl builds for Linux architectures: `mips`, `mipsle`, `mips64`, and `mips64le` (hard float and soft float).
- (Commercial feature) Sensu now automatically applies the `sensu.io/managed_by` label to resources created via `sensuctl create` for use in the `sensuctl prune` alpha feature.

IMPROVEMENTS:

- (Commercial feature) The health endpoint now includes PostgreSQL health information.
- Resource metadata now includes the `created_by` field, which Sensu automatically populates with the name of the user who created or last updated each resource.
- The agent now discovers entity libc type, VM system, VM role, and cloud provider.
- System type now includes the `float_type` field, which stores the float type the system is using (hard float or soft float).
- The Bonsai client now logs at the `debug` level rather than the `info` level.
- The store can now create wrapped resources.
- Tessen now collects the type of store used for events (`etcd` or `postgres`) and logs numbers of authentication providers, secrets, and secrets providers. Tessen data helps us understand how we can improve Sensu, and all Tessen transmissions are logged locally for complete transparency.

FIXES:

- Fixed a bug where `event.Check.State` was not set for events passing through the pipeline or written to the event log.
- Fixed a bug that allowed the agent to connect to a backend using a nonexistent namespace.
- Fixed a bug that allowed `subscriptions` to be empty strings.
- Corrected the HTTP status codes for unauthenticated and permission denied errors in the REST API.
- Fixed a bug where check history was incorrectly formed when using the PostgreSQL event store.

5.18.1 release notes

March 10, 2020 — The latest release of Sensu Go, version 5.18.1, is now available for download. This release fixes bugs that caused SQL migration failure on PostgreSQL 12, nil pointer panic due to OICD login, and sensu-backend restart upon agent disconnection. It also includes a reliability improvement — a change to use the gRPC client rather than the embedded etcd client.

Read the upgrade guide to upgrade Sensu to version 5.18.1.

FIXES:

- (Commercial feature) Fixed a bug that caused SQL migrations to fail on PostgreSQL 12.
- (Commercial feature) Fixed a bug where OIDC login could result in a nil pointer panic.
- Changed to using the gRPC client (rather than the embedded etcd client) to improve reliability and avoid nil pointer panics triggered by shutting down the embedded etcd client.
- The Sensu backend no longer hangs indefinitely if a file lock for the asset manager cannot be obtained. Instead, the backend returns an error after 60 seconds.
- Fixed a bug that caused sensu-backend to restart when agents disconnected.
- Fixed a bug where the backend would panic on some 32-bit systems.

5.18.0 release notes

February 25, 2020 — The latest release of Sensu Go, version 5.18.0, is now available for download. This release delivers a number of improvements to the overall Sensu Go experience. From automatic proxy entity creation to unique Sensu event IDs, it's now much easier to use and troubleshoot your monitoring event pipelines! If you're working behind an HTTP proxy, you can now manage remote Sensu Go clusters, as sensuctl now honors proxy environment variables (for example, HTTPS_PROXY). This release also includes a number of fixes for usability bugs, making for the most polished release of Sensu Go yet, so go ahead and give it a download!

Read the upgrade guide to upgrade Sensu to version 5.18.0.

IMPROVEMENTS:

- The `event.entity.entity_class` value now defaults to `proxy` for `POST /events` requests.
- If you use the core/v2/events API to create a new event with an entity that does not already exist, the sensu-backend will automatically create a proxy entity when the event is published.
- Sensuctl now accepts Bonsai asset versions that include a prefix with the letter `v` (for example, `v1.2.0`).
- The /version API now retrieves the Sensu agent version for the Sensu instance.
- Log messages now indicate which filter dropped an event.
- Sensu now reads and writes `initializationKey` to and from EtcdRoot, with legacy support (read-only) as a fallback.
- Sensu will now check for an HTTP response other than `200 OK` response when fetching assets.

- Updated Go version from 1.13.5 to 1.13.7.

FIXES:

- (Commercial feature) Label selectors and field selectors now accept single and double quotes to identify strings.
- Fixed a bug that prevented wrapped resources from having their namespaces set by the default sensuctl configuration.
- Fixed a bug that prevented API response filtering from working properly for core/v2/silenced API endpoints.
- Improved event payload validation for the core/v2/events API so that events that do not match the URL parameters on the `/events/:entity/:check` endpoint are rejected.
- Sensuctl now supports the `http_proxy`, `https_proxy`, and `no_proxy` environment variables.
- The `auth/test` endpoint now returns the correct error messages.

5.17.2 release notes

February 19, 2020 — The latest release of Sensu Go, version 5.17.2, is now available for download. This release fixes a bug that could prevent commercial features from working after internal restart.

Read the upgrade guide to upgrade Sensu to version 5.17.2.

FIXES:

- Fixed a bug that could cause commercial HTTP routes to fail to initialize after an internal restart, preventing commercial features from working.

5.17.1 release notes

January 31, 2020 — The latest release of Sensu Go, version 5.17.1, is now available for download. This release fixes a web UI issue that cleared selected filters when sorting an event list and a bug that prevented certain `.tar` assets from extracting.It also includes sensuctl configuration improvements.

Read the upgrade guide to upgrade Sensu to version 5.17.1.

IMPROVEMENTS:

- Asset names may now include capital letters.
- Running the `sensuctl configure` command now resets the sensuctl cluster configuration.
- When you use `--trusted-ca-file` to configure sensuctl, it now detects and saves the absolute file path in the cluster configuration.

FIXES:

- (Commercial feature) When a silencing entry expires or is removed, it is also removed from the silences view in the web UI.
- Fixed a bug that prevented `.tar` assets from extracting if they contain hardlinked files.
- In the web UI, sorting an event list view no longer clears the selected filters.

5.17.0 release notes

January 28, 2020 — The latest release of Sensu Go, version 5.17.0, is now available for download. This is a significant release, with new features, improvements, and fixes!We're ecstatic to announce the release of secrets management, which eliminates the need to expose sensitive information in your Sensu configuration.When a Sensu component such as a check or handler requires a secret (like a username or password), Sensu will be able to fetch that information from one or more external secrets providers (for example, HashiCorp Vault) and provide it to the Sensu component via temporary environment variables.Secrets management allows you to move secrets out of your Sensu configuration, giving you the ability to safely and confidently share your Sensu configurations with your fellow Sensu users!This release also includes per-entity keepalive event handler configuration, a sought-after feature for users who have migrated from Sensu 1.x to Sensu Go.

Read the upgrade guide to upgrade Sensu to version 5.17.0.

NEW FEATURES:

- (Commercial feature) Added HTTP API for secrets management, with Sensu's `Env` secrets provider and support for HashiCorp Vault secrets management. The secrets provider resource is implemented for checks, mutators, and handlers.
- Added the `keepalive-handlers` agent configuration option to specify the keepalive handlers to use for an entity's events.

IMPROVEMENTS:

- (Commercial feature) Upgraded the size of the events auto-incremented ID in the PostgreSQL store to a 64-bit variant, which allows you to store many more events and avoids exhausting the sequence.
- (Commercial feature) Initialization via `sensu-backend init` is now implemented for Docker.
- (Commercial feature) UPN binding support has been re-introduced via the `default_upn_domain` configuration attribute.
- In the web UI, labels that contain URLs are now clickable links.
- Added `event.entity.name` as a supported field for the `fieldSelector` query parameter.
- In the web UI, users with implicit permissions to a namespace can now display resources within that namespace.
- Explicit access to namespaces can only be granted via cluster-wide RBAC resources.
- You can now omit the namespace from an event in `HTTP POST /events` requests.
- Added support for the `--format` flag in the sensuctl command list subcommand.

FIXES:

- (Commercial feature) Fixed a bug where the event check state was not present when using the PostgreSQL event store.
- (Commercial feature) Agent TLS authentication does not require a license.
- Fixed a memory leak in the entity cache.
- Fixed a bug that prevented `sensuctl entity delete` from returning an error when attempting to delete a non-existent entity.
- In the web UI, fixed a bug that duplicated event history in the event timeline chart.
- `sensuctl command` assets installed via Bonsai now use the `sensuctl` namespace.
- Fixed a bug where failing check TTL events could occur if keepalive failures had already occurred.

5.16.1 release notes

December 18, 2019 — The latest release of Sensu Go, version 5.16.1, is now available for download. This release fixes a performance regression that caused API latency to scale linearly as the number of connected agents increased and includes a change to display the `sensu_go_events_processed` Prometheus counter by default.

Read the upgrade guide to upgrade Sensu to version 5.16.1.

IMPROVEMENTS

- The `sensu_go_events_processed` Prometheus counter now initializes with the `success` label so the count is always displayed.

FIXES:

- The performance regression introduced in 5.15.0 that caused API latency to scale linearly as the number of connected agents increased is fixed.

5.16.0 release notes

December 16, 2019 — The latest release of Sensu Go, version 5.16.0, is now available for download. This is another important release, with many new features, improvements, and fixes.We introduced an initialization subcommand for **new** installations that allows you to specify an admin username and password instead of using a pre-defined default.We also added new backend configuration options to help you take advantage of etcd auto-discovery features and agent configuration options you can use to define a timeout period for critical and warning keepalive events.

New web UI features include a switcher that makes it easier to switch between namespaces in the dashboard, breadcrumbs on every page, OIDC authentication in the dashboard, a drawer that replaces the app bar to make more room for content, and more.

We also fixed issues with `sensuctl dump` and `sensuctl cluster health`, installing sensuctl commands via Bonsai, and missing namespaces in keepalive events and events created through the agent socket interface.

Read the upgrade guide to upgrade Sensu to version 5.16.0.

IMPORTANT:

- For Debian- and RHEL-family installations, the backend is no longer seeded with a default admin username and password.Users will need to run 'sensu-backend init' on every new installation and specify an admin username and password.

NEW FEATURES:

- (Commercial feature) Users can now authenticate with OIDC in the dashboard.

- (Commercial feature) Label selectors now match the event's check and entity labels.
- Added a new configuration option, `etcd-client-urls`, to use with sensu-backend when it is not operating as an etcd member. The configuration option is also used by the new `sensu-backend init` subcommand.
- Added the 'sensu-backend init' subcommand.
- Added the `etcd-discovery` and `etcd-discovery-srv` configuration options to sensu-backend, which allow users to take advantage of the embedded etcd's auto-discovery features.
- Added the `keepalive-critical-timeout` configuration option to define the time after which a critical keepalive event should be created for an agent and the `keepalive-warning-timeout` configuration option, which is an alias of `keepalive-timeout` for backward compatibility.

IMPROVEMENTS:

- (Commercial feature) The entity limit warning message is now displayed less aggressively and the warning threshold is proportional to the entity limit.
- A new switcher in the web UI makes it easier to switch namespaces in the dashboard. Access the new component from the drawer or with the shortcut ctrl+k. For users who have many namespaces, the switcher now includes fuzzy search and improved keyboard navigation.
- In the web UI, replaced the app bar with an omnipresent drawer to increase the available space for content. Each page also now includes breadcrumbs.
- In the Sensu documentation, links now point to the version of the product being run instead of the latest, which may be helpful when running an older version of Sensu.

FIXES:

- `sensuctl dump` help now shows the correct default value for the format flag.
- Installing sensuctl commands via Bonsai will now check for correct labels before checking if the asset has 1 or more builds.
- Listing assets with no results now returns an empty array.
- Fixed a panic that could occur when creating resources in a namespace that does not exist.
- Fixed an issue where keepalive events and events created through the agent's socket interface could be missing a namespace.
- Fixed an issue that could cause 'sensuctl cluster health' to hang indefinitely.
- (Commercial feature) The `agent.yml.example` file shipped with Sensu Agent for Windows packages now uses DOS-style line endings.

5.15.0 release notes

November 19, 2019 — The latest release of Sensu Go, version 5.15.0, is now available for download. This is a significant release for a number of reasons. The changes to licensing make 100% of Sensu Go's commercial features available for free to all users, up to your first 100 entities! This release also includes the long-awaited cluster federation features, supporting multi-cluster authentication, RBAC policy replication, and a single pane of glass for your Sensu monitoring data! We added support for API keys, making it easy to integrate with the Sensu API (you no longer need to manage JWTs). In addition, the 5.15.0 release includes support for sensu-backend environment variables and bug fixes that improve error logging for mutator execution and flap detection weighting for checks.

Read the upgrade guide to upgrade Sensu to version 5.15.0.

IMPORTANT: Sensu's free entity limit is now 100 entities. All commercial features are available for free in the packaged Sensu Go distribution for up to 100 entities. You will receive a warning when you approach the 100-entity limit (at 75%).

If your Sensu instance includes more than 100 entities, contact us to learn how to upgrade your installation and increase your limit. Read the blog announcement for more information about our usage policy.

NEW FEATURES:

- (Commercial feature) Added support for federation replicators and the federation cluster registration API and the ability to view resources across clusters in the federation in the web UI.
- (Commercial feature) Added MSI and NuGet builds for sensuctl. Also, MSI and NuGet installations now add the bin directory to the system PATH on Windows.
- (Commercial feature) Added HTTP DELETE access for the license management API.
- Added the APIKey resource, with HTTP API support for POST, GET, and DELETE and sensuctl commands to manage the APIKey resource.
- Added support for using API keys for API authentication.
- Added support for sensuctl commands to install, execute, list, and delete commands from Bonsai or a URL.
- Added support for sensu-backend service environment variables.
- Added support for timezones in check `cron` strings.

SECURITY:

- (Commercial feature) Removed support for UPN binding without a binding account or anonymous binding, which allows Sensu to effectively refresh claims during access token renewal.

IMPROVEMENTS:

- You can now use colons and periods in all resource names (except users).

FIXES:

- Added better error logging for mutator execution.
- Fixed the order of flap detection weighting for checks.
- Fixed the pprof server so it only binds to localhost.
- Moved `corev2.BonsaiAsset` to `bonsai.Asset` and moved `corev2.OutdatedBonsaiAsset` to `bonsai.OutdatedAsset`.

5.14.2 release notes

November 4, 2019 — The latest release of Sensu Go, version 5.14.2, is now available for download. This release includes an etcd upgrade, fixes that improve stability and performance, and a Sensu Go package for RHEL 8.

Read the upgrade guide to upgrade Sensu to version 5.14.2.

IMPROVEMENTS:

- Upgraded etcd to 3.3.17.
- Added build package for RHEL 8 (`el/8`).
- Sensu Go now uses serializable event reads, which helps improve performance.

FIXES:

- As a result of upgrading etcd, TLS etcd clients that lose their connection will successfully reconnect when using the `no-embed-etcd` configuration option.
- Check TTL and keepalive switches are now correctly buried when associated events and entities are deleted. As a result, Sensu now uses far fewer leases for check TTLs and keepalives, which improves stability for most deployments.

- Corrected a minor UX issue in interactive filter commands in sensuctl.

5.14.1 release notes

October 16, 2019 — The latest release of Sensu Go, version 5.14.1, is now available for download. This release adds Prometheus gauges for check schedulers and fixes several bugs, including a bug discovered in 5.14.0 that prevented OIDC authentication providers from properly loading on start-up.

Read the upgrade guide to upgrade Sensu to version 5.14.1.

NEW FEATURES:

- Added Prometheus gauges for check schedulers.

FIXES:

- (Commercial feature) Sensuctl will not incorrectly warn of entity limits for unlimited licenses.
- (Commercial feature) `oidc` authentication providers will now properly load on start-up.
- When opening a Bolt database that is already open, `sensu-agent` will not hang indefinitely.
- Running `sensuctl dump` for multiple resource types with the output format as YAML will not result in separators being printed to stdout instead of the specified file.
- Fixed a crash in sensu-backend (panic: send on closed channel).

5.14.0 release notes

October 8, 2019 — The latest release of Sensu Go, version 5.14.0, is now available for download. This release includes feature additions like two new configuration options for backends using embedded etcd and a new SemVer field in entity resources. In addition, this release includes enhanced TLS authentication support and bug fixes that restore check execution after a network error and enable round robin schedule recovery after quorum loss.

Read the upgrade guide to upgrade Sensu to version 5.14.0.

NEW FEATURES:

- The web UI now includes an error dialog option that allows users to wipe the application's

persisted state (rather than having to manually wipe their local/session storage). This can help in the rare case that something in the persisted state is leading to an uncaught exception.

- The web UI now respects the system preference for operating systems with support for selecting a preferred light or dark theme.
- `sensuctl dump` can now list the types of supported resources with `sensuctl dump --types`.
- The entity resource now includes the `sensu_agent_version` field, which reflects the Sensu Semantic Versioning (SemVer) version of the agent entity.
- There are two new advanced configuration options for `sensu-backend` using embedded etcd: `etcd-heartbeat-interval` and `etcd-election-timeout`.

IMPROVEMENTS:

- (Commercial feature) Added support for mutual TLS authentication between agents and backends.
- (Commercial feature) Added support for CRL URLs for mTLS authentication.
- (Commercial feature) Support agent TLS authentication is usable with the sensu-backend.
- In the web UI, feedback is directed to Discourse rather than the GitHub repository's Issues page to facilitate discussion about feature requests.
- In the web UI, when a user lands on a page inside a namespace that no longer exists or they do not have access to, the drawer opens to that namespace switcher to help clarify next steps.
- Updated Go version from 1.12.3 to 1.13.1.

FIXES:

- (Commercial feature) `sensuctl` on Windows can now create Postgres resources.
- (Commercial feature) Fixed a bug that resulted in event metrics being ignored when using the Postgres store.
- Fixed a bug that caused checks to stop executing after a network error.
- Fixed a bug that prevented `sensuctl create` with stdin from working.
- Splayed proxy checks are executed every interval (instead of every interval + interval * splay_coverage).
- Proxy entity labels and annotations are now redacted in the web UI as expected.
- Fixed a bug in the ring that prevented round robin schedules from recovering after quorum loss.

- Updated web UI so that unauthorized errors emitted while creating silences or resolving events are caught and a notification is presented to communicate what occurred.
- Web UI does not report internal errors when a user attempts to queue an ad hoc check for a keepalive.
- Fixed a bug in the web UI that may have prevented users with appropriate roles from resolving events, queuing checks, and creating silenced entries.
- Asset builds are not separated into several assets unless the the tabular format is used in `sensuctl asset list`.
- The 'flag accessed but not defined' error is corrected in `sensuctl asset outdated`.

5.13.2 release notes

September 19, 2019 — The latest release of Sensu Go, version 5.13.2, is now available for download. This is a stability release that fixes a bug for users who have the PostgreSQL event store enabled.

Read the upgrade guide to upgrade Sensu to version 5.13.2.

FIXES:

- Metrics handlers now correctly receive metric points when the postgresql event store is enabled.

5.13.1 release notes

September 10, 2019 — The latest release of Sensu Go, version 5.13.1, is now available for download. This is a stability release with bug fixes for multi-build asset definitions causing a panic when no matching filters are found.

Read the upgrade guide to upgrade Sensu to version 5.13.1.

FIXES:

- Multi-build asset definitions with no matching filters will no longer cause a panic.
- Fixed the `oidc` authentication provider resource.

5.13.0 release notes

September 9, 2019 — The latest release of Sensu Go, version 5.13.0, is now available for download. This is one of the most user-friendly releases yet! Sensuctl now integrates with Bonsai, the Sensu asset hub, making it easier than ever to fetch and use countless Sensu monitoring plugins and integrations. Additionally, sensuctl now supports loading resource configuration files (for example, checks) from directories and URLs. But that's not all! Sensuctl now provides a subcommand for exporting its configuration and API tokens to your shell environment. Use sensuctl to provide cURL and custom scripts with fresh API access information!

Read the upgrade guide to upgrade Sensu to version 5.13.0.

NEW FEATURES:

- Sensuctl now integrates with Bonsai, the Sensu asset hub. Run a single sensuctl command to add an asset to your Sensu cluster (for example, `sensuctl asset add sensu/sensu-pagerduty-handler:1.1.0`). Check for outdated assets (new releases available) with the `outdated` subcommand (for example, `sensuctl asset outdated`).
- Sensuctl now supports the `env` subcommand for exporting sensuctl configuration and API tokens to your shell environment (for example, `eval $(sensuctl env)`).
- Sensuctl now supports loading multiple resource configuration files (for example, checks and handlers) from directories! Sensuctl can also load a file using a URL (for example, `sensuctl create -r -f ./checks` and `sensuctl create -f https://my.blog.ca/sensu-go/check.yaml`).

FIXES:

- Sensuctl interactive check create and update modes now have `none` for the metric output format as the first highlighted option instead of `nagios-perfdata`.
- Fixed a bug where silences would not expire on event resolution.

5.12.0 release notes

August 26, 2019 — The latest release of Sensu Go, version 5.12.0, is now available for download. There are some exciting feature additions in this release, including the ability to output resources to a file from sensuctl and more granular control of check and check hook execution with an agent allow list. Additionally, this release includes the ability to delete assets and more stability fixes around watcher functionality.

Read the upgrade guide to upgrade Sensu to version 5.12.0.

IMPORTANT:

Due to changes in the release process, Sensu binary-only archives are now named following the pattern `sensu-go_5.12.0_$OS_$ARCH.tar.gz`, where `$OS` is the operating system name and `$ARCH` is the CPU architecture. These archives include all files in the top level directory. Read the installation guide for the latest download links.

NEW FEATURES:

- Operators can now authenticate to Sensu via OpenID Direct Connect (OIDC) using sensuctl. Read the authentication documentation for details.
- Added sensu-agent and sensuctl binary builds for FreeBSD.
- Added sensuctl dump command to output resources to a file or stdout, making it easier to back up your Sensu backends.
- Agents can now be configured with a list of executables that are allowed to run as check and hook commands. Read the agent reference for more information.

IMPROVEMENTS:

- Assets now support defining multiple builds, reducing the number of individual assets needed to cover disparate platforms in your infrastructure.
- (Commercial feature) Namespaces listed in both the web UI and sensuctl are now limited to the namespace to which the user has access.
- Hooks now support the use of assets.
- The event.check.name field has been added as a supported field selector.
- Both the API and sensuctl can now be used to delete assets.
- The use of ProtoBuf serialization/deserialization over WebSocket can now be negotiated between agent and backend.
- Web UI performance has been improved for deployments with many events and entities.
- The resource caches can now rebuild themselves in case of failures.
- Event and entity resources can now be created via the API without an explicit namespace. The system will refer to the namespace in the request URL.
- Event and entity resources can now be created via the API using the POST verb.

SECURITY:

- To prevent writing sensitive data to logs, the backend no longer logs decoded check result and keepalive payloads.

FIXES:

- Tabular display of filters via sensuctl now displays `&&` or `||` as appropriate for inclusive and exclusive filters, respectively.
- Requesting events from the `GET /events/:entity` API endpoint now returns events only for the specified entity.
- Running sensuctl config view without configuration no longer causes a crash.
- Creating an entity via sensuctl with the `--interactive` flag now prompts for the entity name when it is not provided on the command line.
- Check hooks with `stdin: true` now receive actual event data on stdin instead of an empty event.
- Some issues with check scheduling and updating have been fixed by refactoring the backend's watcher implementation.

KNOWN ISSUES:

- Authentication via OIDC is not yet supported in the web UI.
- Deleting an asset will not remove references to said asset. It is the operator's responsibility to remove the asset from the runtime_assets field of the check, hook, filter, mutator, or handler.
- Deleting an asset will not remove the tarball or downloaded files from disk. It is the operator's responsibility to clear the asset cache if necessary.

5.11.1 release notes

July 18, 2019 — The latest release of Sensu Go, version 5.11.1, is now available for download. This is a stability release with bug fixes for UPN format binding token renewal and addition of agent heartbeats and configurable WebSocket connection negotiation.

Read the upgrade guide to upgrade Sensu to version 5.11.1.

FIXES:

- Fixed access token renewal when UPN format binding was enabled.

- The agent now sends heartbeats to the backend to detect network failures and reconnect more quickly.
- The default handshake timeout for the WebSocket connection negotiation was lowered from 45 to 15 seconds and is now configurable.

5.11.0 release notes

July 10, 2019 — The latest release of Sensu Go, version 5.11.0, is now available for download. There are some exciting feature additions in this release, including the ability to delete resources from sensuctl and manage filter and mutator resources in the web UI. Additionally, this release includes bug fixes for proxy checks and enhanced performance tuning for the PostgreSQL event store.

Read the upgrade guide to upgrade Sensu to version 5.11.0.

NEW FEATURES:

- The Sensu web UI now includes a filters page that displays available event filters and filter configuration.
- (Commercial feature) Manage your Sensu event filters from your browser: Sensu's web UI now supports creating, editing, and deleting filters.
- The Sensu web UI now includes a mutators page that displays available mutators and mutator configuration.
- (Commercial feature) Manage your Sensu mutators from your browser: Sensu's web UI now supports creating, editing, and deleting mutators.
- Sensuctl now includes the `sensuctl delete` command, letting you use resource definitions to delete resources from Sensu in the same way as `sensuctl create`. Read the sensuctl reference for more information.
- Assets now include a `headers` attribute to include HTTP headers in requests to retrieve assets, allowing you to access secured assets. Read the asset reference for examples.
- Sensu agents now support the `disable-assets` configuration option, allowing you to disable asset retrieval for individual agents. Read the agent reference for examples.
- Sensu binary-only distributions are now available as zip files.

IMPROVEMENTS:

- (Commercial feature) The Active Directory authentication provider now supports the `default_upn_domain` attribute, letting you appended a domain to a username when a domain

is not specified during login.

- (Commercial feature) The Active Directory authentication provider now supports the `include_nested_groups` attribute, letting you search nested groups instead of just the top-level groups of which a user is a member.
- The `sensuctl config view` command now returns the currently configured username. Read the sensuctl reference for examples.
- The Sensu API now returns the `201 Created` response code for POST and PUT requests instead of `204 No Content`.
- The Sensu backend now provides advanced configuration options for buffer size and worker count of keepalives, events, and pipelines.
- Sensu Go now supports Debian 10. For a complete list of supported platforms, visit the platforms page.

FIXES:

- The web UI now returns an error when attempting to create a duplicate check or handler.
- Silenced entries are now retrieved from the cache when determining whether an event is silenced.
- The Sensu API now returns an error when trying to delete an entity that does not exist.
- The agent WebSocket connection now performs basic authorization.
- The core/v2/events API now correctly applies the current timestamp by default, fixing a regression in 5.10.0.
- Multiple nested set handlers are now flagged correctly, fixing an issue in which they were flagged as deeply nested.
- Round robin proxy checks now execute as expected in the event of updated entities.
- The Sensu backend now avoids situations of high CPU usage in the event that watchers enter a tight loop.
- Due to incompatibility with the Go programming language, Sensu is incompatible with RHEL 5. As a result, RHEL 5 has been removed as a supported platform for all versions of Sensu Go.

5.10.2 release notes

June 27, 2019 — The latest release of Sensu Go, version 5.10.2, is now available for download. This is a stability release with a bug fix for expired licenses.

Read the upgrade guide to upgrade Sensu to version 5.10.2.

FIXES:

- Sensu now handles expired licenses as expected.

5.10.1 release notes

June 25, 2019 — The latest release of Sensu Go, version 5.10.1, is now available for download. This is a stability release with key bug fixes for proxy checks and entity deletion.

Read the upgrade guide to upgrade Sensu to version 5.10.1.

FIXES:

- The proxy_requests entity_attributes are now all considered when matching entities.
- Events are now removed when their corresponding entity is deleted.

5.10.0 release notes

June 19, 2019 — The latest release of Sensu Go, version 5.10.0, is now available for download. There are some exciting feature additions in this release, including the ability to perform advanced filtering in the web UI and use PostgreSQL as a scalable event store. This release also includes key bug fixes, most notably for high CPU usage.

Read the upgrade guide to upgrade Sensu to version 5.10.0.

NEW FEATURES:

- (Commercial feature) The Sensu web UI now includes fast, predictive filtering for viewing checks, entities, events, handlers, and silences, including the ability to filter based on custom labels. Select the filter bar and start building custom views using suggested attributes and values. For more information, read the web UI docs.
- Free Sensu instances can now delete entities in the web UI entities page. Read the web UI docs to get started using the Sensu web UI.
- (Commercial feature) Sensu now supports using an external PostgreSQL instance for event storage in place of etcd. PostgreSQL can handle significantly higher volumes of Sensu events,

letting you scale Sensu beyond etcd's storage limits.Read the datastore reference for more information.

- Sensu now includes a cluster ID API endpoint and `sensuctl cluster id` command to return the unique Sensu cluster ID.Read the core/v2/cluster API endpoint docs for more information.

IMPROVEMENTS:

- The `sensuctl create` command now supports specifying the namespace for a group of resources at the time of creation, allowing you to replicate resources across namespaces without manual editing.Read the sensuctl reference for more information and usage examples.
- Sensu cluster roles can now include permissions to manage your Sensu license using the `license` resource type.Read the RBAC reference to create a cluster role.
- The web UI now displays up to 100,000 events and entities on the homepage.

FIXES:

- Sensu now optimizes scheduling for proxy checks, solving an issue with high CPU usage when evaluating proxy entity attributes.
- The Sensu API now validates resource namespaces and types in request bodies to ensure RBAC permissions are enforced.
- Check `state` and `total_state_change` attributes now update as expected based on check history.
- Incident and entity links in the web UI homepage now navigate to the correct views.
- The web UI now displays non-standard cron statements correctly (for example, `@weekly`).
- On sign-in, the web UI now ensures that users are directed to a valid namespace.
- In the web UI, code block scrollbars now display only when necessary.
- The web UI now displays the handler `timeout` attribute correctly.
- When editing resources, the web UI now fetches the latest resource prior to editing.
- The web UI now handles array values correctly when creating and editing resources.

5.9.0 release notes

May 28, 2019 — The latest release of Sensu Go, version 5.9.0, is now available for download.There are some exciting feature additions in this release, including the ability to log raw events to a file (commercial feature) and view event handlers in the web UI.

Read the upgrade guide to upgrade Sensu to version 5.9.0. If you're upgrading a Sensu cluster from 5.7.0 or earlier, read the instructions for upgrading a Sensu cluster from 5.7.0 or earlier to 5.8.0 or later.

NEW FEATURES:

- The Sensu web UI now includes a handlers page that displays available event handlers and handler configuration. Read the web UI docs to get started using the Sensu web UI.
- (Commercial feature) Manage your Sensu event handlers from your browser: Sensu's web UI now supports creating, editing, and deleting handlers. Read the web UI docs to get started using the Sensu web UI.
- (Commercial feature) Sensu now supports event logging to a file using the `event-log-file` and `event-log-buffer-size` configuration options. You can use this event log file as an input source for your favorite data lake solution. Read the backend reference for more information.

IMPROVEMENTS:

- The Sensu web UI now includes simpler, more efficient filtering in place of filtering using Sensu query expressions.
- Sensu packages are now available for Ubuntu 19.04 (Disco Dingo). Review the supported platforms page for a complete list of Sensu's supported platforms and the installation guide to install Sensu packages for Ubuntu.

FIXES:

- The `occurrences` and `occurrences_watermark` event attributes now increment as expected, giving you useful information about recent events. Read the events reference for an in-depth discussion of these attributes.
- The `/silenced/subscriptions/:subscription` and `/silenced/checks/:check` API endpoints now return silences by check or subscription.
- Sensu now handles errors when seeding initial data, avoiding a panic state.

5.8.0 release notes

May 22, 2019 — The latest release of Sensu Go, version 5.8.0, is now available for download. This is mainly a stability release with bug fixes and performance improvements. Additionally, we have added support for configurable etcd cipher suites.

Read the upgrade guide to upgrade Sensu to version 5.8.0.

IMPORTANT:

- To upgrade to Sensu Go 5.8.0, Sensu clusters with multiple backend nodes must be shut down during the upgrade process.Read the upgrade guide for more information.

IMPROVEMENTS:

- The sensuctl command line tool now supports the `--chunk-size` flag to help you handle large datasets.Read the sensuctl reference for more information.
- Sensu backends now support the `etcd-cipher-suites` configuration option, letting you specify the cipher suites that can be used with etcd TLS configuration.Read the backend reference for more information.
- The Sensu API now includes the /version API, returning version information for your Sensu instance.Review the API docs for more information.
- Tessen now collects the numbers of events processed and resources created, giving us better insight into how we can improve Sensu.As always, all Tessen transmissions are logged for complete transparency.Read the Tessen reference for more information.
- Sensu licenses now include the entity limit attached to your Sensu licensing package.Read the license management docs to learn more about entity limits.
- Sensu backends now perform better at scale using increased worker pool sizes for events and keepalives.
- The maximum size of the etcd database and etcd requests is now configurable using the `etcd-quota-backend-bytes` and `etcd-max-request-bytes` backend configuration options. These are advanced configuration options requiring familiarly with etcd.Use with caution.Read the backend reference for more information.
- Most Sensu resources now use ProtoBuf serialization in etcd.

FIXES:

- Events produced by checks now execute the correct number of write operations to etcd.
- API pagination tokens for the core/v2/users and core/v2/namespaces API endpoints now work as expected.
- Keepalive events for deleted and deregistered entities are now cleaned up as expected.

KNOWN ISSUES:

- Auth tokens may not be purged from etcd, resulting in a possible impact to performance.

5.7.0 release notes

May 9, 2019 — The latest release of Sensu Go, version 5.7.0, is now available for download. This is mainly a stability release with bug fixes. Additionally, we have added support for Windows packages and updated our usage policy.

Read the upgrade guide to upgrade Sensu to version 5.7.0.

IMPROVEMENTS:

- The Sensu agent for Windows is now available as an MSI package, making it easier to install and operate. Read the installation guide and the agent reference to get started.

FIXES:

- Sensu now enforces resource separation between namespaces sharing a similar prefix.
- The `sensuctl cluster` commands now output correctly in JSON and wrapped JSON formats.
- The API now returns an error message if label and field selectors are used without a license.

5.6.0 release notes

April 30, 2019 — The latest release of Sensu Go, version 5.6.0, is now available for download. We have added some exciting new features in this release, including API filtering and the ability to create and manage checks through the web UI with the presence of a valid license key.

Read the upgrade guide to upgrade Sensu to version 5.6.0.

NEW FEATURES:

- (Commercial feature) Manage your Sensu checks from your browser: Sensu's web user interface now supports creating, editing, and deleting checks. Read the web UI docs to get started using the Sensu web UI.
- (Commercial feature) The Sensu web UI now includes an option to delete entities.
- (Commercial feature) Sensu now supports resource filtering in the Sensu API and sensuctl

command line tool. Filter events using custom labels and resource attributes, such as event status and check subscriptions. Review the API docs and sensuctl reference for usage examples.

IMPROVEMENTS:

- (Commercial feature) Sensu's LDAP and Active Directory integrations now support mutual authentication using the `trusted_ca_file`, `client_cert_file`, and `client_key_file` attributes. Read the guide to configuring an authentication provider for more information.
- (Commercial feature) Sensu's LDAP and Active Directory integrations now support connecting to an authentication provider using anonymous binding. Read the LDAP and Active Directory binding configuration docs to learn more.
- the /health API response now includes the cluster ID.
- The `sensuctl cluster health` and `sensuctl cluster member-list` commands now include the cluster ID in tabular format.

FIXES:

- You can now configure labels and annotations for Sensu agents using command line flags. For example: `sensu-agent start --label example_key="example value"`. Read the agent reference for more examples.
- The Sensu web UI now displays the correct checkbox state when no resources are present.

5.5.1 release notes

April 17, 2019 — The latest release of Sensu Go, version 5.5.1, is now available for download. This is a stability release with key bug fixes, including addressing an issue with backend CPU utilization. Additionally, we have added support for honoring the source attribute for events received via agent socket.

Read the upgrade guide to upgrade Sensu to version 5.5.1.

IMPROVEMENTS:

- Sensu agents now support annotations (non-identifying metadata) that help people or external tools interacting with Sensu. Read the agent reference to add annotations in the agent configuration file.
- The agent socket event format now supports the `source` attribute to create a proxy entity.

- Sensu 5.5.1 is built with Go version 1.12.3.

FIXES:

- Backends now reinstate etcd watchers in the event of a watcher failure, fixing an issue causing high CPU usage in some components.

5.5.0 release notes

April 4, 2019 — The latest release of Sensu Go, version 5.5.0, is now available for download. This release has some key bug fixes and additions, including the introduction of Tessen into Sensu Go. For more information, read Sean Porter's blog post on Tessen.

Read the upgrade guide to upgrade Sensu to version 5.5.0.

NEW FEATURES:

- Tessen, the Sensu call-home service, is now enabled by default in Sensu backends. Read the Tessen docs to learn about the data that Tessen collects.

IMPROVEMENTS:

- Sensu now includes more verbose check logging to indicate when a proxy request matches an entity according to its entity attributes.

FIXES:

- The Sensu web UI now displays silences created by LDAP users.
- The web UI now uses a secondary text color for quick-navigation buttons.

5.4.0 release notes

March 27, 2019 — The latest release of Sensu Go, version 5.4.0, is now available for download. This release has some very exciting feature additions, including the introduction of our new homepage. It also includes support for API pagination to handle large datasets more efficiently and agent buffering for robustness in lower-connectivity situations, along with key bug fixes.

Read the upgrade guide to upgrade Sensu to version 5.4.0.

NEW FEATURES:

- The Sensu dashboard now includes a homepage designed to highlight the most important monitoring data, giving you instant insight into the state of your infrastructure.Read the web UI docs for a preview.
- The Sensu API now supports pagination using the `limit` and `continue` query parameters, letting you limit your API responses to a maximum number of objects and making it easier to handle large datasets.Read the API overview for more information.
- Sensu now surfaces internal metrics using the /metrics API.Read the /metrics API documentation for more information.

IMPROVEMENTS:

- Sensu now lets you specify a separate TLS certificate and key to secure the dashboard.Read the backend reference to configure the `dashboard-cert-file` and `dashboard-key-file` options, and check out the guide to securing Sensu for the complete guide to making your Sensu instance production-ready.
- The Sensu agent events API now queues events before sending them to the backend, making the agent events API more robust and preventing data loss in the event of a loss of connection with the backend or agent shutdown.Read the agent reference for more information.

FIXES:

- The backend now processes events without persisting metrics to etcd.
- The core/v2/events API POST and PUT endpoints now add the current timestamp to new events by default.
- The core/v2/users API endpoints now return a 404 response code if a username cannot be found.
- The sensuctl command line tool now correctly accepts global flags when passed after a subcommand flag (for example, `--format yaml --namespace development`).
- The `sensuctl handler delete` and `sensuctl filter delete` commands now correctly delete resources from the currently configured namespace.
- The agent now terminates consistently on SIGTERM and SIGINT.
- In the event of a loss of connection with the backend, the agent now attempts to reconnect to any backends specified in its configuration.
- The dashboard now handles cases in which the creator of a silence is inaccessible.
- The dashboard event details page now displays “-” in the command field if no command is

associated with the event.

5.3.0 release notes

March 11, 2019 — The latest release of Sensu Go, version 5.3.0, is now available for download. This release has some very exciting feature additions and key bug fixes. Active Directory can be configured as an authentication provider (commercial feature). Additionally, round robin scheduling has been fully re-implemented and is available for use.

Read the upgrade guide to upgrade Sensu to version 5.3.0.

NEW FEATURES:

- Round robin check scheduling lets you distribute check executions evenly over a group of Sensu agents. To enable round robin scheduling, set the `round_robin` check attribute to `true`. Read the checks reference for more information.
- Sensu now provides commercial support for using Microsoft Active Directory as an external authentication provider. Read the authentication guide to configure Active Directory, and check out the getting started guide for more information about commercial features.
- The dashboard now features offline state detection and displays an alert banner if the dashboard loses connection to the backend.

IMPROVEMENTS:

- The agent socket event format now supports the `handlers` attribute, giving you the ability to send socket events to a Sensu pipeline. Read the agent reference to learn more about creating and handling monitoring events using the agent socket.
- Assets now feature improved download performance using buffered I/O.
- The sensuctl CLI now uses a 15-second timeout period when connecting to the Sensu backend.
- The dashboard now includes expandable configuration details sections on the check and entity pages. You can now use the dashboard to review check details like command, subscriptions, and scheduling as well as entity details like platform, IP address, and hostname.

SECURITY:

- Sensu Go 5.3.0 fixes all known TLS vulnerabilities affecting the backend, including increasing the minimum supported TLS version to 1.2 and removing all ciphers except those with perfect

forward secrecy.

- Sensu now enforces uniform TLS configuration for all three backend components: `apid`, `agentd`, and `dashboardd`.
- The backend no longer requires the `trusted-ca-file` configuration option when using TLS.
- The backend no longer loads server TLS configuration for the HTTP client.

FIXES:

- Sensu can now download assets with download times of more than 30 seconds without timing out.
- The agent now communicates entity subscriptions to the backend in the correct format.
- Sensu no longer includes the `edition` configuration attribute or header.
- DNS resolution in Alpine Linux containers now uses the built-in Go resolver instead of the glibc resolver.
- The `sensuctl user list` command can now output `yaml` and `wrapped-json` formats when used with the `--format` flag.
- The dashboard check details page now displays long commands correctly.
- The dashboard check details page now displays the `timeout` attribute correctly.

5.2.1 release notes

February 11, 2019 — The latest release of Sensu Go, version 5.2.1, is now available for download. This is a stability release with a key bug fix for proxy check functionality.

Read the upgrade guide to upgrade Sensu to version 5.2.1.

FIXES:

- Sensu agents now execute checks for proxy entities at the expected interval.

5.2.0 release notes

February 7, 2019 — The latest release of Sensu Go, version 5.2.0, is now available for download. This release has a ton of exciting content, including the availability of our first enterprise-only features. For

more details on these features, read the blog post about Sensu Go 5.2.0.Release 5.2.0 also has some key improvements and fixes: we added support for self-signed CA certificates for sensuctl, check output truncation, and the ability to manage silencing from the event details page in our web UI, to name a few.

Read the upgrade guide to upgrade Sensu to version 5.2.0.

IMPORTANT:

- Due to changes in the release process, Sensu binary-only archives are now named following the pattern `sensu-enterprise-go_5.2.0_$OS_$ARCH.tar.gz`, where `$OS` is the operating system name and `$ARCH` is the CPU architecture.These archives include all files in the top-level directory.Read the installation guide for the latest download links.

NEW FEATURES:

- Our first enterprise-only features for Sensu Go: LDAP authentication, the Sensu ServiceNow handler, and the Sensu JIRA handler.Read the getting started guide.
- Sensu now provides the option to limit check output size or to drop check outputs following metric extraction. Read the checks reference for more information.

IMPROVEMENTS:

- Sensu now includes support for Debian 8 and 9.Read the installation guide to install Sensu for Debian.
- Sensu's binary-only distribution for Linux is now available for `arm64`, `armv5`, `armv6`, `armv7`, and `386` in addition to `amd64`.Read the installation guide for download links.
- The Sensu dashboard now provides the ability to silence and unsilence events from the Events page.
- The Sensu dashboard Entity page now displays the platform version and deregistration configuration.
- Sensuctl now supports TLS configuration options, allowing you to use a self-signed certificate without adding it to the operating system's CA store, either by explicitly trusting the signer or by disabling TLS hostname verification.Read the sensuctl reference for more information.
- sensuctl now provides action-specific confirmation messages, like `Created`, `Deleted`, and `Updated`.

FIXES:

- Check TTL failure events now persist through cluster member failures and cluster restarts.

- The Sensu backend now correctly handles errors for missing keepalive events.
- Token-substituted values are now omitted from event data to protect sensitive information.
- Sensu now correctly processes keepalive and check TTL states after entity deletion.
- Sensuctl can now run `sensuctl version` without being configured.
- When disabling users, sensuctl now provides the correct prompt for the action.

5.1.1 release notes

January 24, 2019 — The latest patch release of Sensu Go, version 5.1.1, is now available for download. This release includes some key fixes and improvements, including refactored keepalive functionality with increased reliability. Additionally, based on community feedback, we have added support for the Sensu agent and sensuctl for 32-bit Windows systems.

Read the upgrade guide to upgrade Sensu to version 5.1.1.

NEW FEATURES:

- Sensu now includes a sensuctl command and API endpoint to test user credentials. Read the access control reference and API docs for more information.

IMPROVEMENTS:

- The Sensu agent and sensuctl tool are now available for 32-bit Windows. Read the installation guide for instructions.
- Keepalive events now include an output attribute specifying the entity name and time last sent.
- The Sensu backend includes refactored authentication and licensing to support future enterprise features.

SECURITY:

- Sensu 5.1.1 is built with Go version 1.11.5. Go 1.11.5 addresses a security vulnerability that affects TLS handshakes and JWT tokens. Read the CVE for more information.

FIXES:

- Keepalive events now continue to execute after a Sensu cluster restarts.
-

- When requested, on-demand check executions now correctly retrieve asset dependencies.
- Checks now maintain a consistent execution schedule after updates to the check definition.
- Proxy check request errors now include the check name and namespace.
- When encountering an invalid line during metric extraction, Sensu now logs an error and continues extraction.
- Sensuctl now returns an error when attempting to delete a non-existent entity.
- Sensuctl now removes the temporary file it creates when executing the `sensuctl edit` command.
- The Sensu dashboard now recovers from errors correctly when shutting down.
- The Sensu dashboard includes better visibility for buttons and menus in the dark theme.

5.1.0 release notes

December 19, 2018 — The latest release of Sensu Go, version 5.1.0, is now available for download. This release includes an important change to the Sensu backend state directory as well as support for Ubuntu 14.04 and some key bug fixes.

Read the upgrade guide to upgrade Sensu to version 5.1.0.

IMPORTANT:

> ***NOTE**: This applies only to Sensu backend binaries downloaded from `s3-us-west-2.amazonaws.com/sensu.io/sensu-go`, not to Sensu RPM or DEB packages.*

- For Sensu backend binaries, the default `state-dir` is now `/var/lib/sensu/sensu-backend` instead of `/var/lib/sensu`. To upgrade your Sensu backend binary to 5.1.0, make sure your `/etc/sensu/backend.yml` configuration file specifies a `state-dir`. Read the upgrade guide for more information.

NEW FEATURES:

- Sensu agents now include `trusted-ca-file` and `insecure-skip-tls-verify` configuration options, giving you more flexibility with certificates when connecting agents to the backend over TLS.

IMPROVEMENTS:

- Sensu now includes support for Ubuntu 14.04.

FIXES:

- The Sensu backend now successfully connects to an external etcd cluster.
- SysVinit scripts for the Sensu agent and backend now include correct run and log paths.
- Once created, keepalive alerts and check TTL failure events now continue to occur until a successful event is observed.
- When querying for an empty list of assets, sensuctl and the Sensu API now return an empty array instead of null.
- The sensuctl `create` command now successfully creates hooks when provided with the correct definition.
- The Sensu dashboard now renders status icons correctly in Firefox.

5.0.1 release notes

December 12, 2018 — Sensu Go 5.0.1 includes our top bug fixes following last week's general availability release.

Read the upgrade guide to upgrade Sensu to version 5.0.1.

FIXED:

- The Sensu backend can now successfully connect to an external etcd cluster.
- The Sensu dashboard now sorts silences in ascending order, correctly displays status values, and reduces shuffling in the event list.
- Sensu agents on Windows now execute command arguments correctly.
- Sensu agents now correctly include environment variables when executing checks.
- Command arguments are no longer escaped on Windows.
- Sensu backend environments now include handler and mutator execution requests.

5.0.0 release notes

December 5, 2018 — We're excited to announce the general availability release of Sensu Go! Sensu Go is the flexible monitoring event pipeline written in Go and designed for container-based and hybrid-cloud infrastructures. Check out the Sensu blog for more information about Sensu Go and version 5.0.

For a complete list of changes from Beta 8-1, review the Sensu Go changelog. This page will be the official home for the Sensu Go changelog and release notes.

To get started with Sensu Go:

- Install Sensu Go.
- Get started monitoring server resources.

Get started with Sensu

Sensu Go is the flexible observability pipeline designed for container-based and multi-cloud infrastructures.

Sensu is available as packages, Docker images, and binary-only distributions. You can install the commercial distribution or build Sensu from source.

Install the commercial distribution of Sensu Go

Sensu's supported platforms include Debian- and RHEL-family distributions and Windows.

- **Install Sensu Go** with a commercial package and get started for free
- Learn about Sensu's commercial features — all commercial features are available for free in the packaged Sensu Go distribution up to an entity limit of 100
- Find the Sensu architecture that best meets your needs
- Discover, configure, and install monitoring and observability integrations in the Sensu Catalog and explore hundreds of dynamic runtime assets for deploying plugins in Bonsai, the Sensu asset hub
- Migrate from Sensu Core and Sensu Enterprise to Sensu Go

Learn Sensu

Watch this video for a 10-minute introduction to Sensu Go:

We recommend these resources for learning more about Sensu:

- Follow the self-guided Sensu Go Workshop and create your first observability pipeline
- Try a live demo of the Sensu web UI
- Sign up for our step-by-step Learn Sensu email course
- Join the Sensu Community Forum on Discourse

Explore monitoring at scale with Sensu Go

Sensu offers support packages for Sensu Go as well as commercial licenses designed for monitoring at scale.

- Contact the sales team for a personalized demo and free trial of commercial features at scale
- Activate your Sensu commercial license

Build Sensu from source (OSS)

Sensu Go's core is open source software, freely available under an MIT License.

- Compare OSS and commercial features
- Visit Sensu Go on GitHub
- Build from source

Supported platforms and distributions

Sensu is available as packages, Docker images, and binary-only distributions. We recommend installing Sensu with one of our supported packages, Docker images, or configuration management integrations. Sensu downloads are provided under the Sensu commercial license.

Supported packages

This page lists supported packages for the most common platforms. Supported packages are available from sensu/stable on packagecloud and the Sensu downloads page.

***NOTE**: The sensu/stable repository on packagecloud includes packages for every platform Sensu supports, in addition to packages for the common platforms listed on this page.*

Sensu backend

	RHEL family 6, 7, 8, 9	Debian 8, 9, 10, 11	Ubuntu 14.04 16.04, 18.04, 18.10 19.04, 19.10, 20.04 22.04
`amd64`	✅	✅	✅
`arm64`	✅	✅	✅
`ppc64le`	✅	✅	✅

Sensu agent

	RHEL family 6, 7, 8, 9	Debian 8, 9, 10, 11	Ubuntu 14.04 16.04	Windows 7 and later	Windows Server 2008 R2

			18.04 18.10 19.04 19.10 20.04 22.04		and later
`amd64`	✅	✅	✅	✅	✅
`386`	✅	✅	✅	✅	✅
`armv5` `armv6` `armv7`	✅	✅	✅		
`ppc64le`	✅	✅	✅		
`s390x`	✅	✅	✅		

Sensuctl command line tool

	RHEL family 6, 7, 8, 9	Debian 8, 9, 10, 11	Ubuntu 14.04 16.04 18.04 18.10 19.04 19.10 20.04 22.04	Windows 7 and later	Windows Server 2008 R2 and later
`amd64`	✅	✅	✅	✅	✅
`386`	✅	✅	✅	✅	✅
`armv5` `armv6` `armv7`	✅	✅	✅		
`ppc64le`	✅	✅	✅		

`s390x`	✅	✅	✅

Docker images

Docker images that contain the Sensu backend and Sensu agent are available for Linux-based containers.

Image Name	Base
sensu/sensu	Alpine Linux
sensu/sensu-rhel	Red Hat Enterprise Linux

Binary-only distributions

Sensu binary-only distributions are available in `.zip` and `.tar.gz` formats.

Platform	Architectures
Linux	`386` `amd64` `arm64` `armv5` `armv6` `armv7` `MIPS` `MIPS LE` `MIPS 64` `MIPS 64 LE` `ppc64le` `s390x`
Windows	`386` `amd64`
macOS	`amd64` `amd64 CGO` `arm64`
FreeBSD	`386` `amd64` `armv5` `armv6` `armv7`
Solaris	`amd64`

Linux

Sensu binary-only distributions for Linux are available for the architectures listed in the table below.

For binary distributions, we support the following Linux kernels:

- 3.1.x and later for `armv5`
- 4.8 and later for `MIPS 64 LE hard float` and `MIPS 64 LE soft float`
- 2.6.23 and later for all other architectures

***NOTE**: The Linux `amd64`, `arm64`, and `ppc64le` binary distributions include the agent, backend, and sensuctl CLI.Binaries for all other Linux architectures include only the Sensu agent and sensuctl CLI.*

Architecture	Formats	Architecture	Formats
`386`	`.tar.gz` | `.zip`	`MIPS LE hard float`	`.tar.gz` | `.zip`
`amd64`	`.tar.gz` | `.zip`	`MIPS LE soft float`	`.tar.gz` | `.zip`
`arm64`	`.tar.gz` | `.zip`	`MIPS 64 hard float`	`.tar.gz` | `.zip`
`armv5`	`.tar.gz` | `.zip`	`MIPS 64 soft float`	`.tar.gz` | `.zip`
`armv6`	`.tar.gz` | `.zip`	`MIPS 64 LE hard float`	`.tar.gz` | `.zip`
`armv7`	`.tar.gz` | `.zip`	`MIPS 64 LE soft float`	`.tar.gz` | `.zip`
`MIPS hard float`	`.tar.gz` | `.zip`	`s390x`	`.tar.gz` | `.zip`
`MIPS soft float`	`.tar.gz` | `.zip`	`ppc64le`	`.tar.gz` | `.zip`

For example, to download Sensu for Linux `amd64` in `tar.gz` format:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_linux_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
sha256sum sensu-go_6.8.2_linux_amd64.tar.gz
```

The result should match the checksum for your platform:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_checksums.txt && cat sensu-go_6.8.2_checksums.txt
```

Federal Information Processing Standard (FIPS) Compliance

Builds that support the Federal Information Processing Standard (FIPS) for Federal Risk and Authorization Management Program (FedRAMP) compliance are available for Linux `amd64`.

Sensu FIPS builds with FIPS-mode configuration options are linked with the FIPS 140-2 validated cryptographic library. You can run Red Hat Enterprise Linux (RHEL) with the FIPS-mode kernel option to enforce FIPS systemwide — Sensu FIPS builds comply with this mode.

Contact Sensu to request builds with FIPS support.

Read Configure Sensu for FIPS compliance to learn more about Sensu's FIPS build, including configuration examples.

Windows

Sensu binary-only distributions for Windows are available for the architectures listed in the table below.

We support Windows 7 and later and Windows Server 2008R2 and later for binary distributions.

> ***NOTE**: The Windows binary distributions include only the Sensu agent and sensuctl CLI.*

Architecture	Formats
`386`	`.tar.gz` | `.zip`
`amd64`	`.tar.gz` | `.zip`

For example, to download Sensu for Windows `amd64` in `zip` format:

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_windows_amd64.zip  -OutFile "$env:userprofile\sensu-
```

```
go_6.8.2_windows_amd64.zip"
```

Generate a SHA-256 checksum for the downloaded artifact:

```
Get-FileHash "$env:userprofile\sensu-go_6.8.2_windows_amd64.zip" -Algorithm SHA256 |
Format-List
```

The result should match (with the exception of capitalization) the checksum for your platform:

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-
go_6.8.2_checksums.txt -OutFile "$env:userprofile\sensu-go_6.8.2_checksums.txt"

Get-Content "$env:userprofile\sensu-go_6.8.2_checksums.txt" | Select-String -Pattern
windows_amd64
```

macOS

Sensu binary-only distributions for macOS are available for the architectures listed in the table below.

We support macOS 10.11 and later for binary distributions.

***NOTE**: The macOS binary distributions include only the Sensu agent and sensuctl CLI.*

Architecture	Formats
`amd64`	`.tar.gz` | `.zip`
`amd64 CGO`	`.tar.gz` | `.zip`
`arm64`	`.tar.gz` | `.zip`

For example, to download Sensu for macOS `amd64` in `tar.gz` format:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_darwin_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
shasum -a 256 sensu-go_6.8.2_darwin_amd64.tar.gz
```

The result should match the checksum for your platform:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_checksums.txt && cat sensu-go_6.8.2_checksums.txt
```

Extract the archive:

```
tar -xvf sensu-go_6.8.2_darwin_amd64.tar.gz
```

Copy the executable into your PATH:

```
sudo cp sensuctl /usr/local/bin/
```

FreeBSD

Sensu binary-only distributions for FreeBSD are available for the architectures listed in the table below.

We support FreeBSD 11.2 and later for binary distributions.

> ***NOTE**: The FreeBSD binary distributions include only the Sensu agent and sensuctl CLI.*

Architecture	Formats
`386`	`.tar.gz` | `.zip`

`amd64`	`.tar.gz` | `.zip`
`armv5`	`.tar.gz` | `.zip`
`armv6`	`.tar.gz` | `.zip`
`armv7`	`.tar.gz` | `.zip`

For example, to download Sensu for FreeBSD `amd64` in `tar.gz` format:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_freebsd_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
sha256sum sensu-go_6.8.2_freebsd_amd64.tar.gz
```

The result should match the checksum for your platform:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_checksums.txt && cat sensu-go_6.8.2_checksums.txt
```

Solaris

Sensu binary-only distributions for Solaris are available for the architectures listed in the table below.

We support Solaris 11 and later (not SPARC) for binary distributions.

***NOTE**: The Solaris binary distributions include only the Sensu agent.*

Architecture	Formats

`amd64` `.tar.gz` | `.zip`

For example, to download Sensu for Solaris `amd64` in `tar.gz` format:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_solaris_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact.

```
sha256sum sensu-go_6.8.2_solaris_amd64.tar.gz
```

The result should match the checksum for your platform.

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go_6.8.2_checksums.txt && cat sensu-go_6.8.2_checksums.txt
```

Legacy systems and other platforms

The Sensu Push utility allows you to execute Sensu checks on legacy systems and other platforms that cannot run the Sensu agent, such as AIX and SPARC Solaris.

You can also use cron to run Sensu checks locally on these systems and forward the results to an upstream Sensu backend or agent via the Sensu API.

Build from source

Sensu Go's core is open source software, freely available under an MIT License.Sensu Go instances built from source do not include commercial features such as the web UI, single sign-on (SSO) authentication, and secrets management.Review the feature comparison matrix to learn more.

To build Sensu Go from source, read the Sensu Go installation instructions on GitHub.To download and run the web UI as a separate component, visit the Sensu Go Web GitHub repository.

Mirror packages

To mirror Sensu Go, follow the packagecloud instructions for YUM and APT repository mirroring. The sensu/stable packagecloud repository hosts packages for every Sensu Go version.

Get started with commercial features

Sensu Go offers commercial features designed for monitoring and observability at scale. All commercial features are available in the official Sensu Go distribution, and you can use them for free up to an entity limit of 100. If you have more than 100 entities, contact the Sensu sales team for a free trial.

In addition to the summary on this page, we describe commercial features in detail throughout the documentation. Watch for this notice to identify commercial features:

> ***COMMERCIAL FEATURE**: Access <feature_name> in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Commercial features in Sensu Go

- **Integrate your Sensu observability pipeline with industry-standard tools** like EC2, Jira, ServiceNow, and Sumo Logic with featured integrations and enterprise-tier dynamic runtime assets. Use the built-in Sensu Plus integration to transmit your observability data to Sumo Logic via the HTTP Logs and Metrics Source.
- **Find, configure, and install integrations directly in your browser** with the Sensu Catalog, our online marketplace for monitoring and observability integrations.
- **Manage resources from your browser**: Use the Sensu web UI to manage events and entities and create, edit, and delete checks, handlers, mutators, silences, and event filters, with a single pane of glass view. Create customized global default settings for page size and theme, page-specific settings for page size, order, and selector, and sign-in messages.
- **Control access with single sign-on (SSO) authentication** using Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).
- **Maintain high-level visibility into the current health of your business services**. Use business service monitoring (BSM) to monitor your system with a top-down approach that produces meaningful alerts, prevents alert fatigue, and helps you focus on your core business services.
- **Use mutual transport layer security (mTLS) authentication** to provide two-way verification of your Sensu agents and backend connections.
- **Protect your sensitive information** with secrets management. Avoid exposing usernames, passwords, and access keys in your Sensu configuration by integrating with HashiCorp Vault

or using Sensu's `Env` secrets provider.

- **Manage your monitoring resources across multiple data centers, cloud regions, or providers** and mirror changes to follower clusters with federation. Federation affords visibility into the health of your infrastructure and services across multiple distinct Sensu instances within a single web UI.
- **Use powerful search capabilities** designed for large installations to search Sensu API responses, sensuctl outputs, and Sensu web UI views using custom labels and a wide range of resource attributes. Build event filter expressions with JavaScript execution functions.
- **Achieve enterprise-scale event handling** for your Sensu instance with a PostgreSQL event store. Access the PostgreSQL event datastore with the same Sensu web UI, API, and sensuctl processes as etcd-stored events. Use Sumo Logic metrics handlers and TCP stream handlers to provide a persistent connection for transmitting Sensu observability metrics.
- **Get enterprise-class support**: Rest assured that with Sensu support, help is available if you need it. Our expert in-house team offers best-in-class support to get you up and running smoothly.

Review a complete comparison of OSS and commercial features.

Contact us for a free trial

Sensu's commercial features are free for your first 100 entities. If your Sensu installation includes more than 100 entities, contact the Sensu sales team for a free trial of commercial features at scale in Sensu Go.

Manage your Sensu account and contact support through account.sensu.io.

Get started with commercial features in Sensu Go

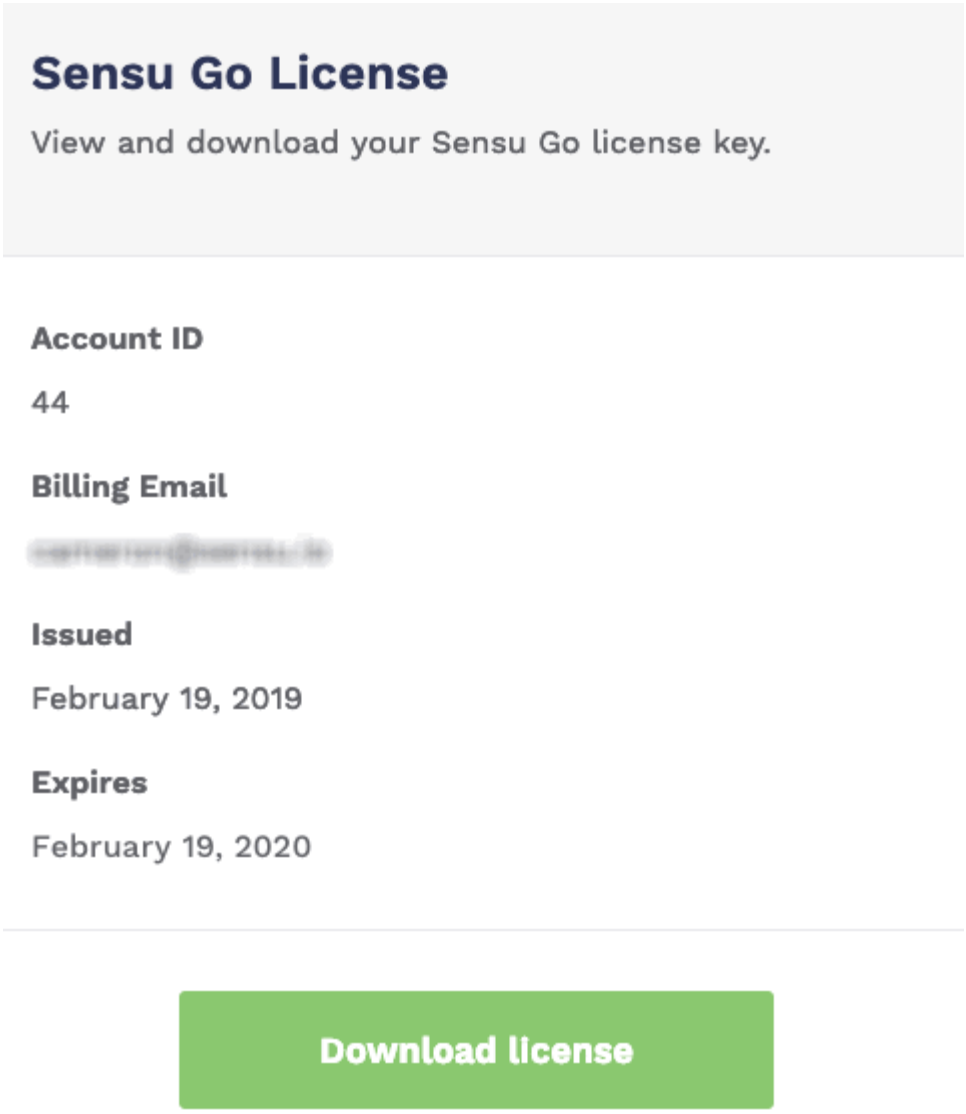
If you haven't already, install the Sensu Go backend, agent, and sensuctl tool and configure sensuctl.

You will need a commercial license if your Sensu installation includes more than 100 entities. To download your commercial license file:

1. Log in to your Sensu account at account.sensu.io.
2. Click **Download license**.

***NOTE**: In some cases, you may need to click **Generate license** before you can download your*

license.



With the license file downloaded, you can use sensuctl to activate your commercial license:

```
sensuctl create --file sensu_license.json
```

***NOTE**: For clustered configurations, you only need to activate your license for one of the backends within the cluster.*

Use sensuctl to view your license details at any time:

```
sensuctl license info
```

Users with permission to create or update licenses can also view license information in the Sensu web UI by pressing `CTRL .` to open the system information modal.

These resources will help you use commercial features in Sensu Go:

- Configure mTLS authentication for the Sensu agent.
- Federate multiple Sensu instances to gain single-pane-of-glass visibility into your infrastructure and services.
- Install plugins with dynamic runtime assets and use our complete catalog of integrations.
- Keep sensitive information like passwords and access tokens private with secrets management.
- Set up and manage single sign-on (SSO) authentication providers: Active Directory (AD), Lightweight Directory Access Protocol (LDAP), and OpenID Connect 1.0 protocol (OIDC).
- Use the web UI for a unified view of your events, entities, and configuration resources along with user-friendly tools, and create customized page views.
- Monitor business services and get high-level visibility into every component in your system.
- Search in the web UI or use powerful response filtering in API requests and sensuctl commands.
- Scale your monitoring and observability with Sensu's enterprise datastore.
- Manage your Sensu commercial license
- Log in to your Sensu account
- Contact Sensu support

Sensu Plus

***COMMERCIAL FEATURE**: Access Sensu Plus in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu Plus is a built-in integration for transmitting your Sensu observability data to the Sumo Logic Continuous Intelligence Platform™ via a Sumo Logic HTTP Logs and Metrics Source (an endpoint for receiving data). In Sumo Logic, you can configure customized interactive dashboards and analytics tools to gain better visibility into your Sensu data — read Introducing Sensu Plus for more information.

If you completed Sumo Logic setup in the Sensu web UI, finish the configuration process to create the Sensu resources you need to start sending observability event data to Sumo Logic.

If you did **not** use the Sensu web UI to set up a Sumo Logic account, follow the manual setup process for Sensu Plus.

Finish configuration after web UI setup

In the Sensu web UI, you already set up a Sumo Logic account and HTTP Logs and Metrics Source endpoint for receiving data. To finish your configuration, you need the **SOURCE URL** for your endpoint, which you can copy from the last page in the web UI dialog.

Follow the steps in this section to create a Sensu handler, pipeline, and check to transmit your Sensu data in Sumo Logic.

Create a handler in Sensu

Create a Sumo Logic metrics handler to send your Sensu observability data to your new Sumo Logic HTTP Logs and Metrics Source. Sumo Logic metrics handlers provide a persistent connection to transmit Sensu observability data, which helps prevent the data bottlenecks you may experience with traditional handlers.

***NOTE**: Sumo Logic metrics handlers only accept metrics events. To send status events, use the Sensu Sumo Logic Handler integration instead.*

For a Sumo Logic metrics handler, the resource definition must use the URL for your HTTP Logs and Metrics Source as the value for the `url` attribute.

Here is an example Sumo Logic Metrics Handler definition.Before you run the command to add this handler, replace the `url` example value with the URL for your Sumo Logic HTTP Logs and Metrics Source:

TEXT

```
cat << EOF | sensuctl create
---
type: SumoLogicMetricsHandler
api_version: pipeline/v1
metadata:
  name: sumo_logic_http_metrics
spec:
  url: "https://collectors.sumologic.com/receiver/v1/http/xxxxxxxx"
  max_connections: 10
  timeout: 10s
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumo_logic_http_metrics"
  },
  "spec": {
    "url": "https://collectors.sumologic.com/receiver/v1/http/xxxxxxxx",
    "max_connections": 10,
    "timeout": "10s"
  }
}
EOF
```

If you prefer, you can configure your Sumo Logic HTTP Logs and Metrics Source URL as a secret with

Sensu's `Env` secrets provider to avoid exposing the URL in your handler definition. This example shows the same definition with the URL referenced as a secret instead:

TEXT

```
cat << EOF | sensuctl create
---
type: SumoLogicMetricsHandler
api_version: pipeline/v1
metadata:
  name: sumo_logic_http_metrics
spec:
  url: $SUMO_LOGIC_SOURCE_URL
  secrets:
  - name: SUMO_LOGIC_SOURCE_URL
    secret: sumologic_metrics_us1
  max_connections: 10
  timeout: 10s
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumo_logic_http_metrics"
  },
  "spec": {
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
        "secret": "sumologic_metrics_us1"
      }
    ],
    "max_connections": 10,
    "timeout": "10s"
  }
}
EOF
```

Configure a pipeline

Sensu pipelines use event filters, mutators, and handlers as the building blocks for event processing workflows. With your handler definition configured, you're ready to create a pipeline with a workflow that references your sumo_logic_http_metrics handler.

To configure event processing via your sumo_logic_http_metrics handler, add this example pipeline definition. This pipeline includes a workflow with your sumo_logic_http_metrics handler, along with Sensu's built-in has_metrics event filter to ensure that the workflow only processes events that contain metrics:

TEXT

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: sensu_to_sumo
spec:
  workflows:
  - name: metrics_to_sumologic
    filters:
    - name: has_metrics
      type: EventFilter
      api_version: core/v2
    handler:
      name: sumo_logic_http_metrics
      type: SumoLogicMetricsHandler
      api_version: pipeline/v1
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "sensu_to_sumo"
  },
  "spec": {
    "workflows": [
      {
        "name": "metrics_to_sumologic",
        "filters": [
          {
            "name": "has_metrics",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "sumo_logic_http_metrics",
          "type": "SumoLogicMetricsHandler",
          "api_version": "pipeline/v1"
        }
      }
    ]
  }
}
EOF
```

Add a Sensu check

Your pipeline resource is now properly configured, but it's not processing any events because no Sensu checks are sending events to it. To get your Sensu observability data flowing through the new pipeline, add the pipeline by reference in at least one check definition.

This example check definition uses the sensu/system-check dynamic runtime asset. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Follow these steps to configure the required system check:

1. Add the sensu/system-check dynamic runtime asset:

```
sensuctl asset add sensu/system-check:0.1.1 -r system-check
```

2. Update at least one Sensu entity to use the `system` subscription. In the following command, replace `<ENTITY_NAME>` with the name of the entity on your system. Then, run:

```
sensuctl entity update <ENTITY_NAME>
```

   - For `Entity Class`, press enter.
   - For `Subscriptions`, type `system` and press enter.

3. Add the following check definition:

TEXT

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: system-check
spec:
  command: system-check
  runtime_assets:
  - system-check
  subscriptions:
  - system
  interval: 10
  timeout: 5
  publish: true
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: sensu_to_sumo
  output_metric_format: prometheus_text
  output_metric_tags:
  - name: entity
    value: "{{ .name }}"
  - name: namespace
    value: "{{ .namespace }}"
  - name: os
    value: "{{ .system.os }}"
  - name: platform
```

```
    value: "{{ .system.platform }}"
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "system-check"
  },
  "spec": {
    "command": "system-check",
    "runtime_assets": [
      "system-check"
    ],
    "subscriptions": [
      "system"
    ],
    "interval": 10,
    "timeout": 5,
    "publish": true,
    "pipelines": [
      {
        "type": "Pipeline",
        "api_version": "core/v2",
        "name": "sensu_to_sumo"
      }
    ],
    "output_metric_format": "prometheus_text",
    "output_metric_tags": [
      {
        "name": "entity",
        "value": "{{ .name }}"
      },
      {
        "name": "namespace",
        "value": "{{ .namespace }}"
      },
      {
```

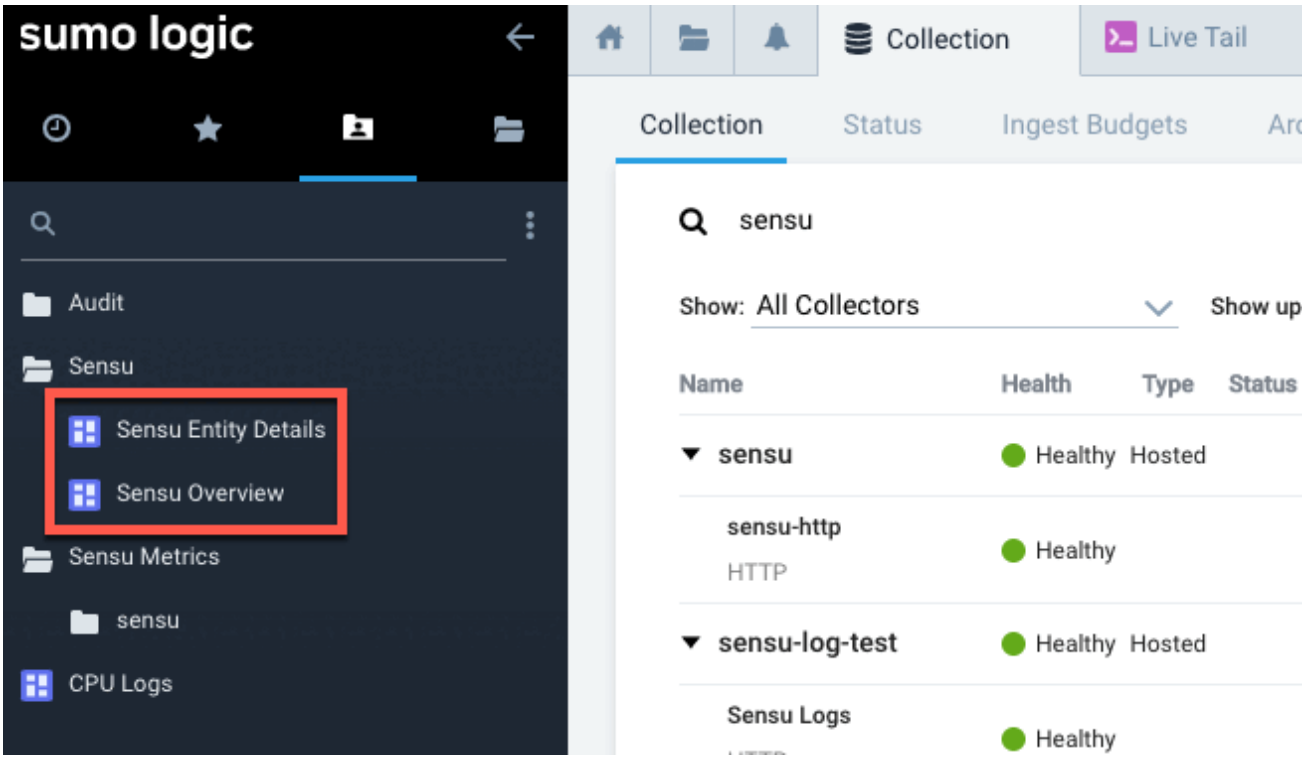
```
          "name": "os",
          "value": "{{ .system.os }}"
        },
        {
          "name": "platform",
          "value": "{{ .system.platform }}"
        }
      ]
    }
}
EOF
```

This check will collect baseline system metrics in Prometheus format for all entities that include the `system` subscription and send the events to Sumo Logic via your sensu_to_sumo pipeline resource.

> ***NOTE**: Sumo Logic metrics handlers only accept metrics events, so you must use a check that produces metrics. If your check produces status events, use the Sensu Sumo Logic Handler integration to create a traditional Sensu handler rather than the Sumo Logic metrics handler.*

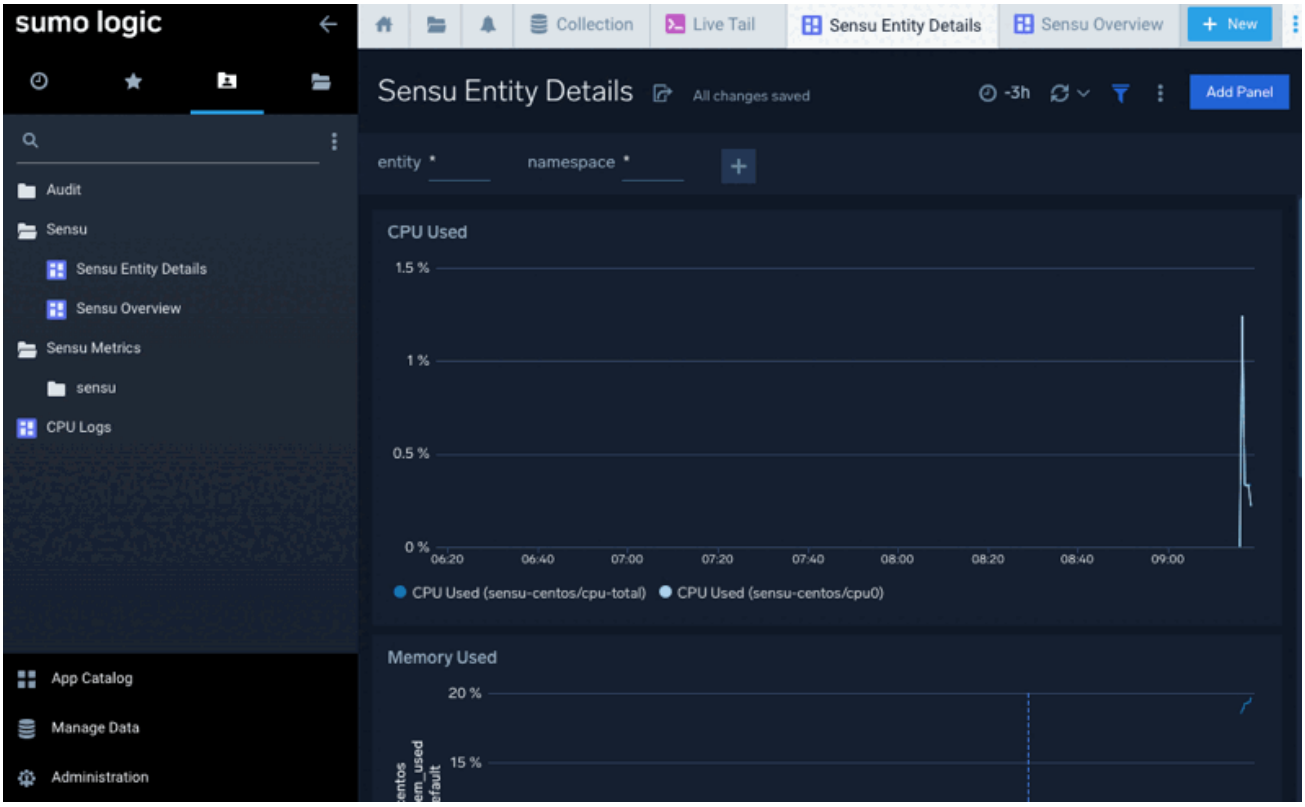
View your Sensu data in Sumo Logic

During the web UI setup process, Sensu added two custom dashboards as a starting point for viewing your observability event data. The two new dashboards will be listed in the Sensu folder in the left-navigation menu:



Click a dashboard name to view your Sensu observability data.

It may take a few moments for your data to appear in Sumo Logic. The Sensu Overview and Sensu Entity Details dashboards will begin to display your data:



Manually set up Sensu Plus

This section explains how to set up Sensu Plus manually, without using the Sensu web UI.

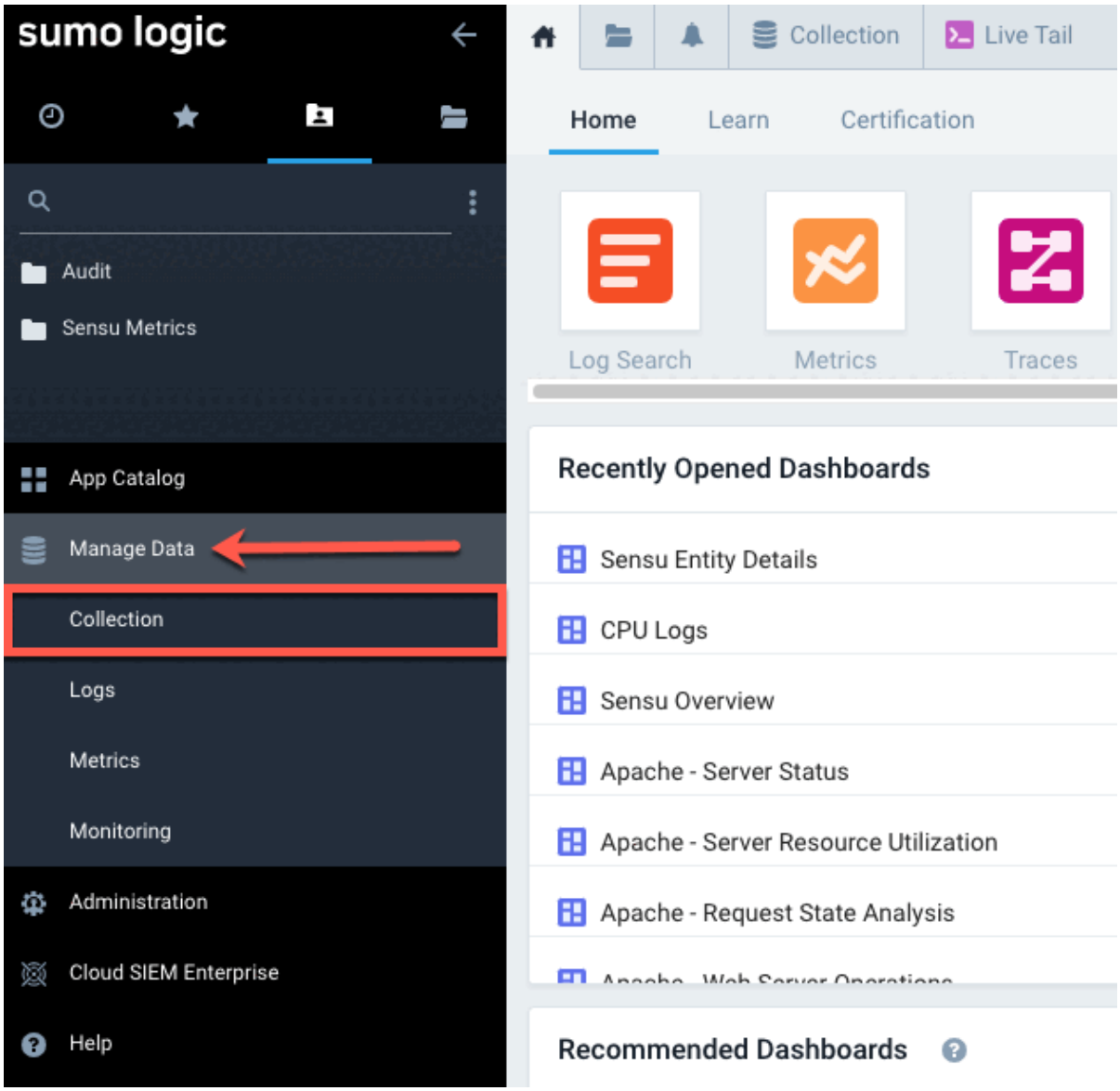
First, create a new Sumo Logic account or log in to your existing account.

Then follow the steps below to create a Sumo Logic HTTP Logs and Metrics Source (an endpoint for receiving data), the Sensu resources that collect and process the data, and two dashboards for viewing your observability event data in Sumo Logic.

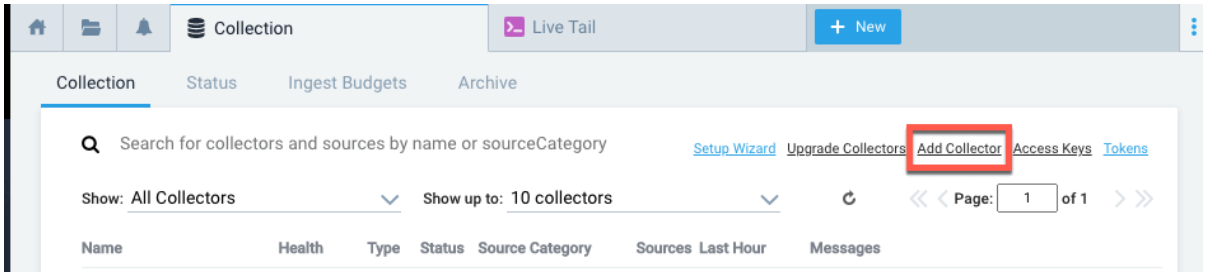
Set up an HTTP Logs and Metrics Source

Create a Sumo Logic HTTP Logs and Metrics Source to collect your Sensu observability data:

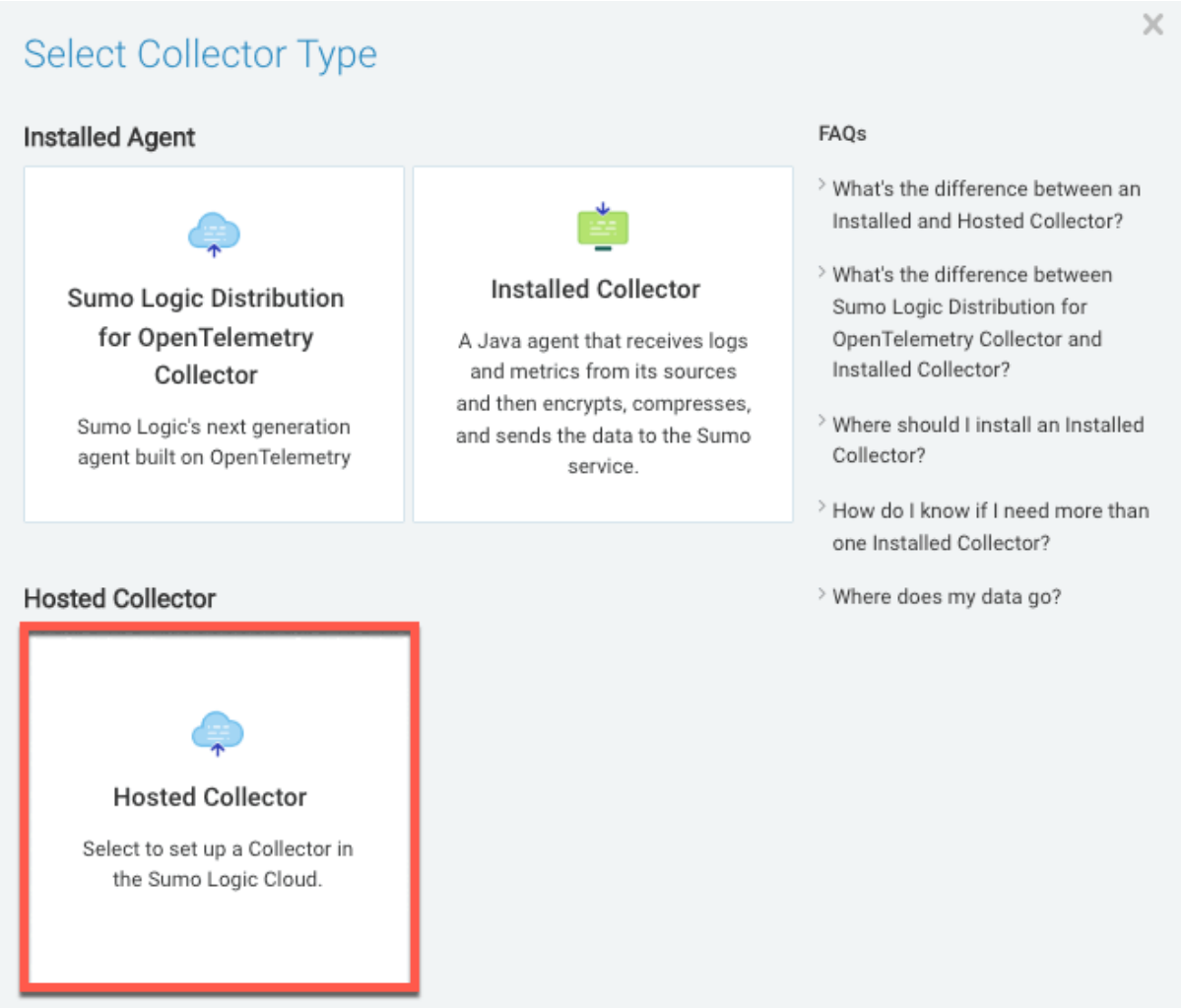
1. In the Sumo Logic left-navigation menu, click **Manage Data** and then **Collection**.



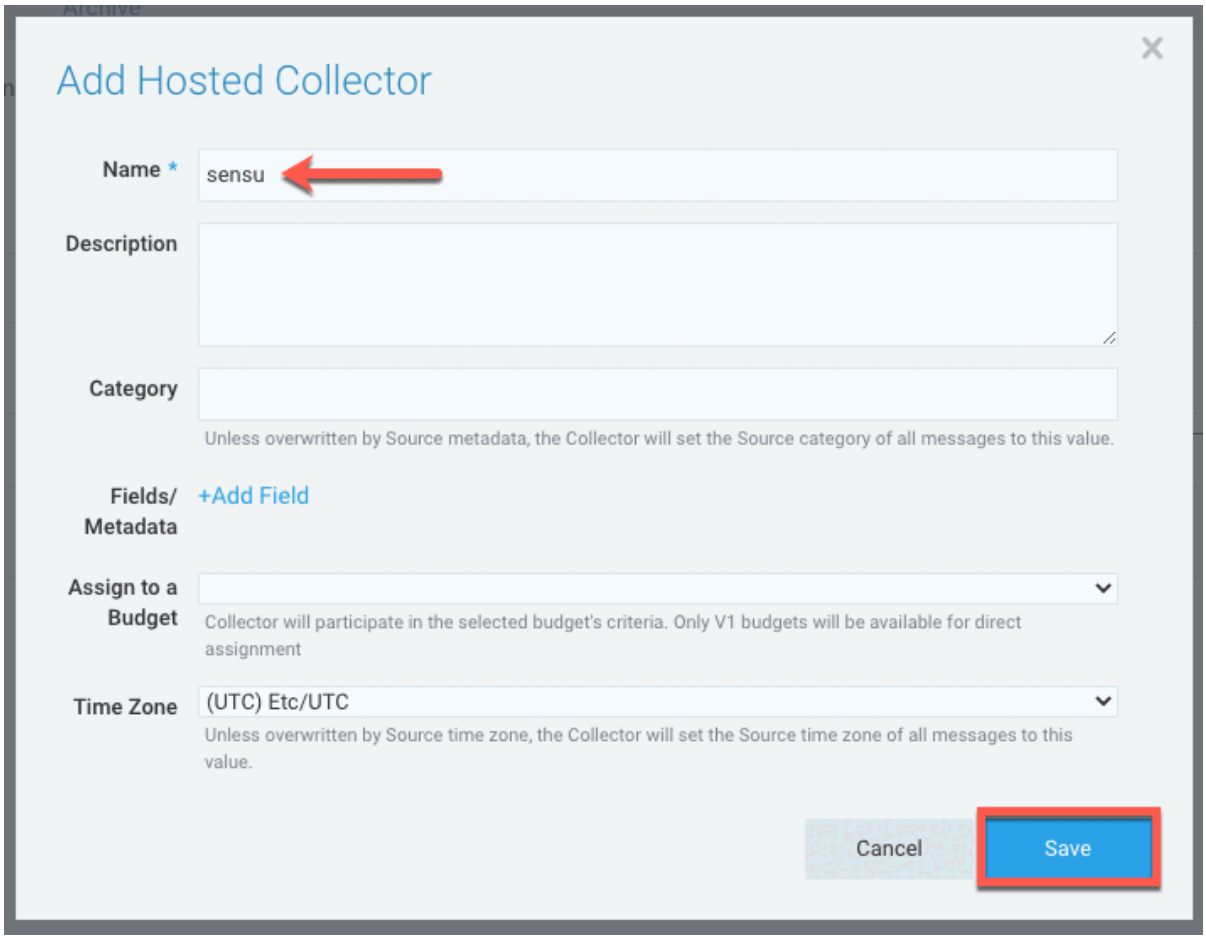
2. At the top-right of the Collection tab, click **Add Collector**.



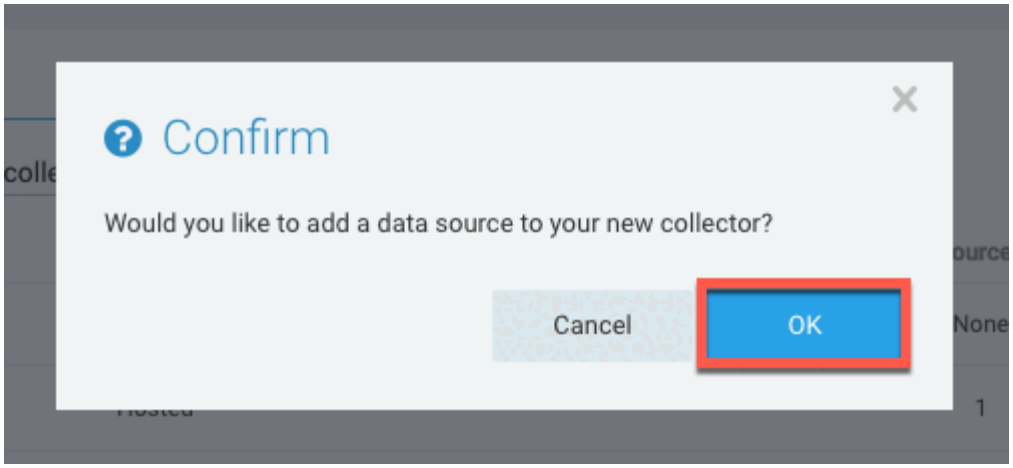
3. In the Click Selector Type modal window, click **Hosted Collector**.



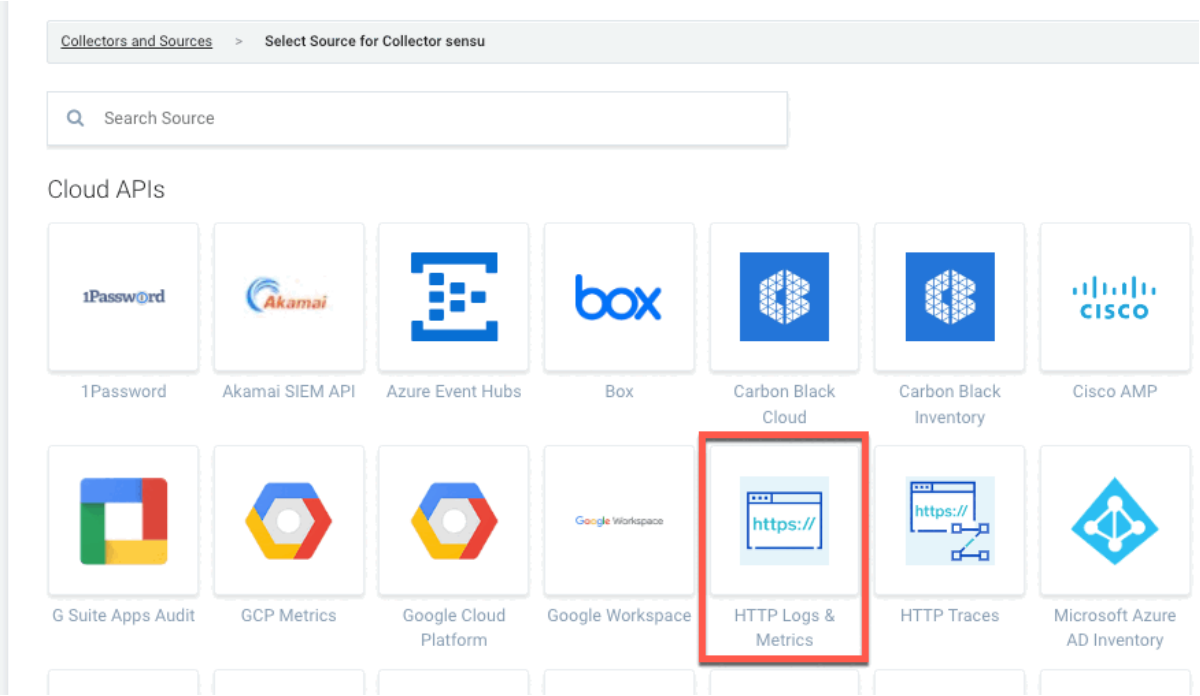
4. In the Add Hosted Collector modal window:
    - Type **sensu** in the Name field.
    - Click **Save**.



5. In the Confirm prompt, click **OK**.

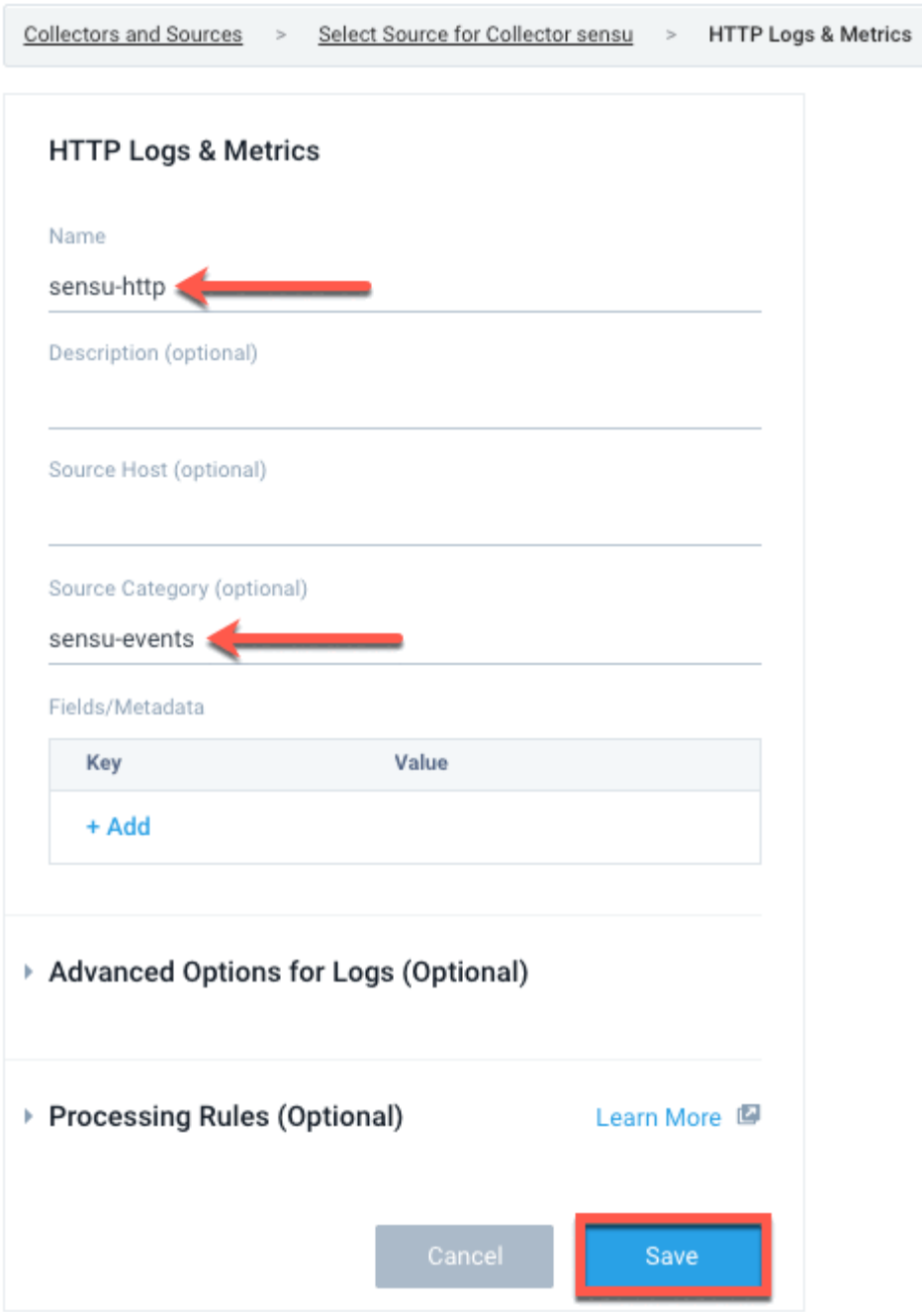


6. Under Cloud APIs, click **HTTP Logs & Metrics**.

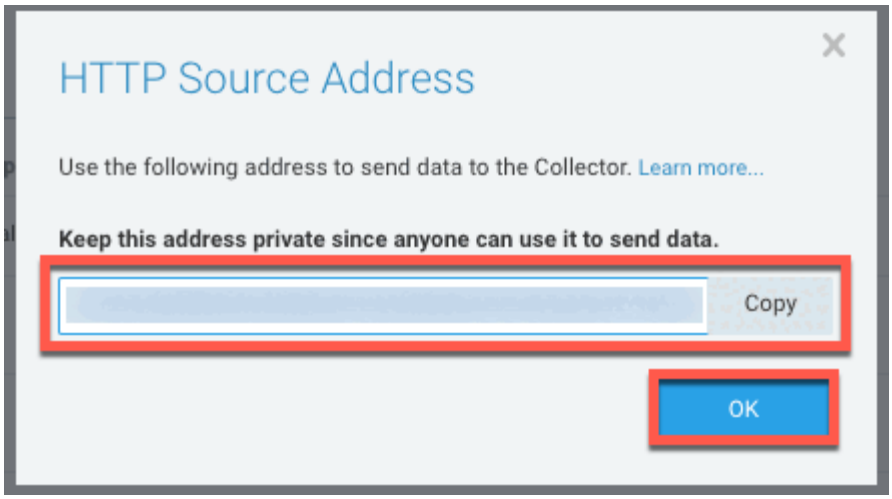


7. In the HTTP Logs & Metrics form:

   - Type **sensu-http** in the Name field.
   - Type **sensu-events** in the Source Category field.
   - Click **Save**.



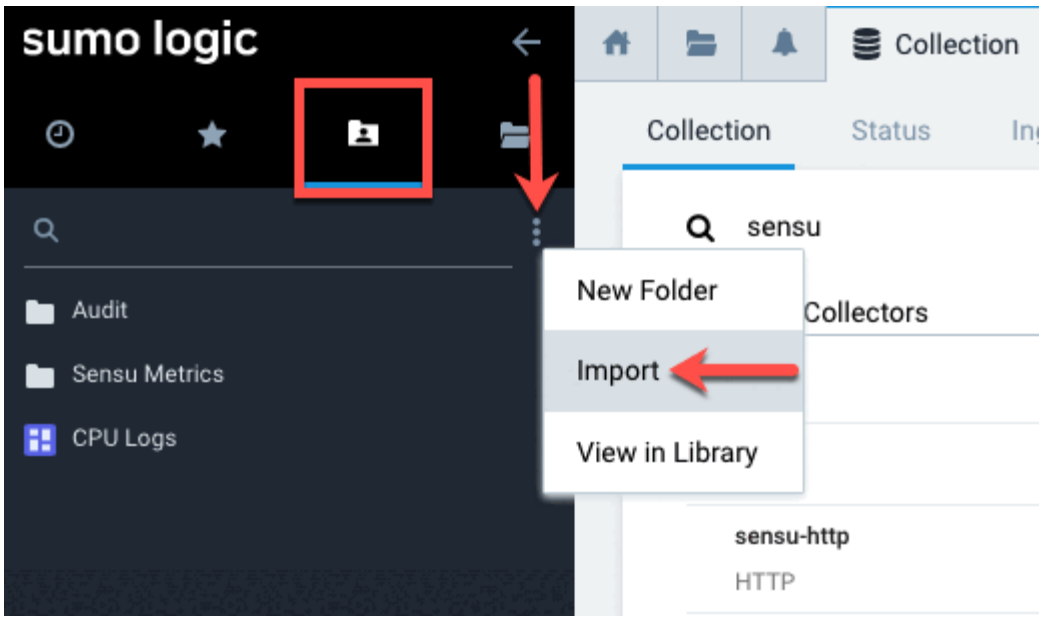
8. In the HTTP Source Address prompt, copy the listed URL and click OK. You will use this URL as the value for the `url` attribute in your Sensu handler definition.



Import Sumo Logic dashboards

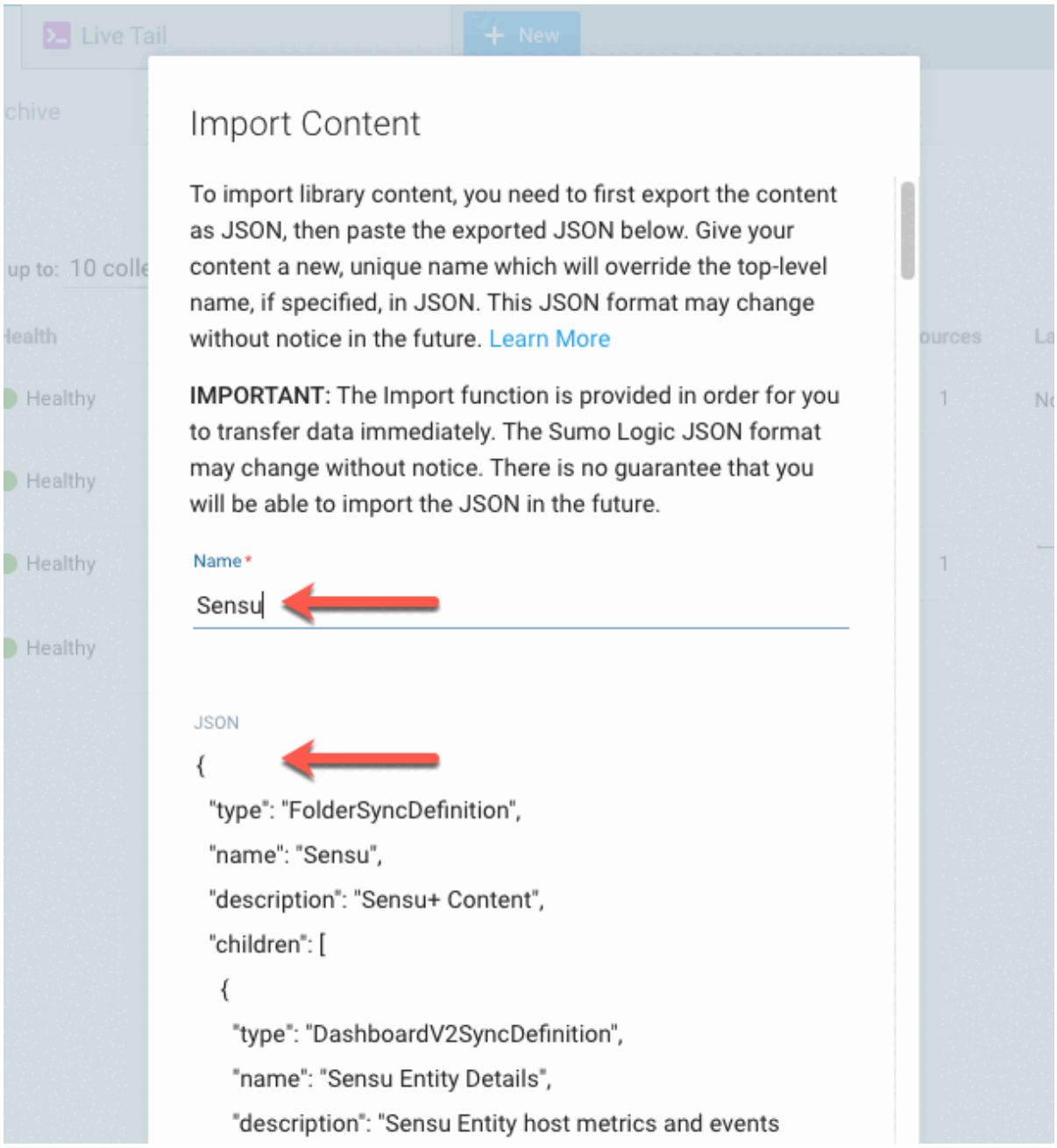
To view your Sensu observability data in Sumo Logic, you can configure Sumo Logic dashboards in any way you wish. As a starting point, follow these instructions to import two dashboards, Sensu Overview and Sensu Entity Details:

1. On your Sumo Logic home page, click the **Personal** tab in the left-navigation menu. Click the options icon for the folder where you want to import your Sensu data and select **Import**.

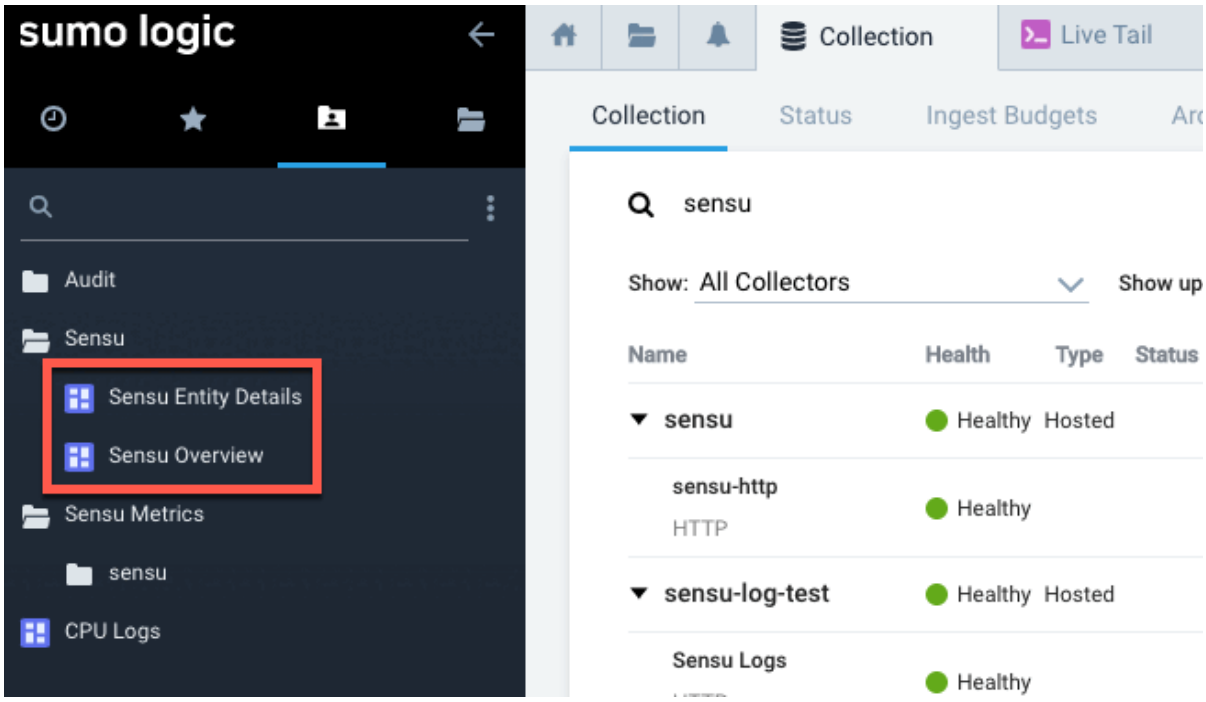


2. In the Import Content modal window:

    - Type “Sensu” in the **Name** field.
    - Copy the dashboard configuration JSON (download) and paste it into the **JSON** field:



3. Scroll to the bottom of the Import Content modal window and click **Import**. The two new dashboards will be listed in the Sensu folder in the left-navigation menu:



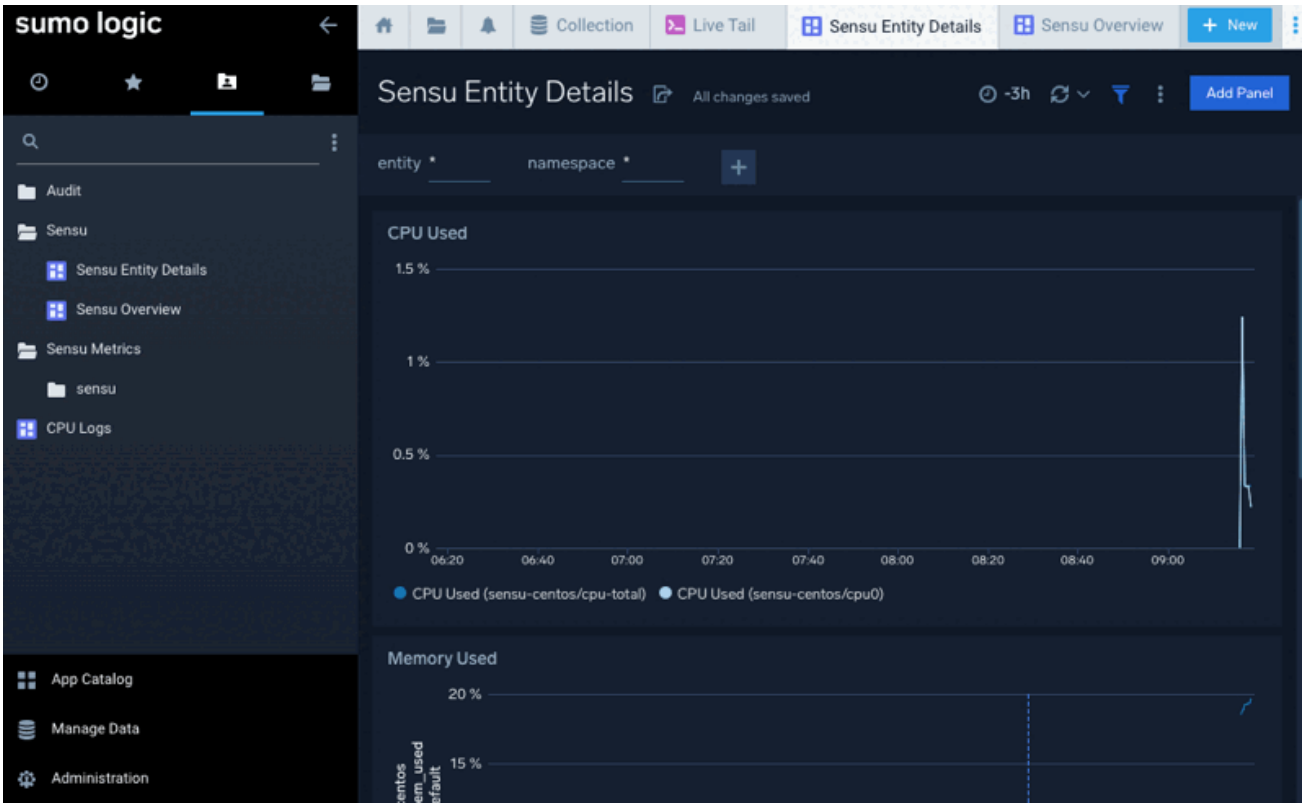
After you create a Sensu handler, pipeline, and check in the next section, you will be able to click a dashboard name to view your Sensu observability data.

Create Sensu resources

With your dashboards set up, you're ready to configure a Sensu handler, pipeline, and check. To create the Sensu resources, follow the same instructions as users who started in the web UI:

- Create a handler in Sensu
- Configure a pipeline
- Add a Sensu check

After you add the check, it may take a few moments for your data to appear in Sumo Logic. The Sensu Overview and Sensu Entity Details dashboards will begin to display your data:


sumo logic
Collection
Live Tail
Sensu Entity Details
Sensu Overview
New
Sensu Entity Details
All changes saved
-3h
Add Panel
entity *
namespace *
Audit
Sensu
Sensu Entity Details
Sensu Overview
Sensu Metrics
sensu
CPU Logs
CPU Used
1.5 %
1 %
0.5 %
0 %
06:20
06:40
07:00
07:20
07:40
08:00
08:20
08:40
09:00
CPU Used (sensu-centos/cpu-total)
CPU Used (sensu-centos/cpu0)
Memory Used
20 %
15 %
App Catalog
Manage Data
Administration

Sensu Observability Pipeline

Sensu's observability pipeline is a flexible, automated tool that gives you visibility into every part of your organization's infrastructure.

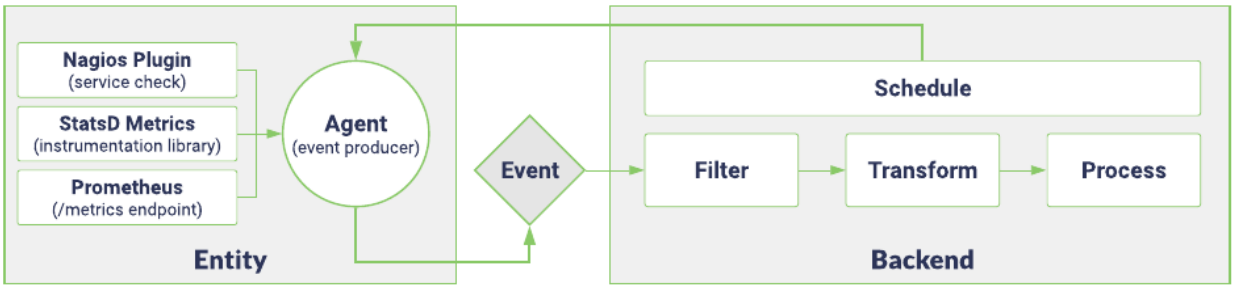
The Sensu agent is a lightweight process that runs on the infrastructure components you want to observe. Each agent is represented in Sensu as an entity. The Sensu backend schedules checks for agents to run on your infrastructure. Agents receive check execution requests based on the agent subscriptions you specify.

The agent runs these checks on your infrastructure to gather observation data about your networking, compute resources, applications, and more. Events contain the observation data that your checks gather, which might include entity status, metrics, or both, depending on your needs and configuration.

The agent sends events to the backend, which filters, transforms, and processes the data in your events with event filters, mutators, and handlers.

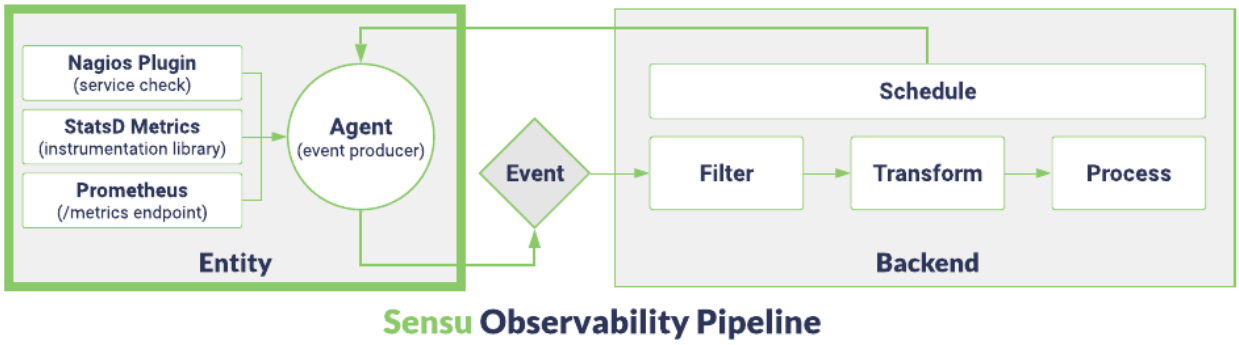
Sensu's observability pipeline delivers contextualized information and deeper insights so you can take targeted actions. For example, Sensu can send entity status data in an email, Slack, or PagerDuty alert and transport metrics to storage in your Graphite, InfluxDB, or Prometheus databases.

Next **or click any element in the pipeline to jump to it.**



Sensu Observability Pipeline

Entities



Sensu Observability Pipeline

Next **or click any element in the pipeline to jump to it.**

An entity represents anything that needs to be observed or monitored, such as a server, container, or network switch, including the full range of infrastructure, runtime, and application types that compose a complete monitoring environment (from server hardware to serverless functions). Sensu calls parts of an infrastructure "entities."

An entity provides the context for observation data in events — what and where the event is from. The check and entity names associated with an event determine the event's uniqueness. Entities can also contain system information like the hostname, operating system, platform, and version.

There are four types of Sensu entities: agent, proxy, service, and backend entities.

Agent entities

Agent entities are monitoring agents that are installed and run on every system that needs to be observed or monitored. The agent entity registers the system with the Sensu backend service, sends keepalive messages (the Sensu heartbeat mechanism), and executes observability checks.

Each entity is a member of one or more `subscriptions`: a list of roles and responsibilities assigned to the agent entity (for example, a webserver or a database). Sensu entities "subscribe" to (or watch for) check requests published by the Sensu backend (via the Sensu transport), execute the corresponding requests locally, and publish the results of the check back to the transport (to be processed by a Sensu backend).

This example shows an agent entity resource definition:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  name: i-424242
spec:
  deregister: false
  deregistration: {}
  entity_class: agent
  last_seen: 0
  sensu_agent_version: 1.0.0
  subscriptions:
  - web
  system:
    cloud_provider: ""
    libc_type: ""
    network:
      interfaces: null
    processes: null
    vm_role: ""
    vm_system: ""
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "i-424242"
  },
  "spec": {
    "deregister": false,
    "deregistration": {
    },
    "entity_class": "agent",
    "last_seen": 0,
    "sensu_agent_version": "1.0.0",
    "subscriptions": [
```

```
      "web"
    ],
    "system": {
      "cloud_provider": "",
      "libc_type": "",
      "network": {
        "interfaces": null
      },
      "processes": null,
      "vm_role": "",
      "vm_system": ""
    }
  }
}
```

Proxy entities

Proxy entities [formerly known as proxy clients or just-in-time (JIT) clients] allow Sensu to monitor external resources on systems where you cannot install a Sensu agent, like a network switch or website.

Proxy entities are dynamically created when an entity does not already exist for a check result. In this case, Sensu uses the `proxy_entity_name` defined in the check to create proxy entities for external resources.

This example shows a proxy entity resource definition:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  labels:
    proxy_type: website
    sensu.io/managed_by: sensuctl
    url: https://docs.sensu.io
  name: sensu-docs
  namespace: default
spec:
```

```
  deregister: false
  deregistration: {}
  entity_class: proxy
  last_seen: 0
  sensu_agent_version: ""
  subscriptions: null
  system:
    cloud_provider: ""
    libc_type: ""
    network:
      interfaces: null
    processes: null
    vm_role: ""
    vm_system: ""
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "labels": {
      "proxy_type": "website",
      "sensu.io/managed_by": "sensuctl",
      "url": "https://docs.sensu.io"
    },
    "name": "sensu-docs",
    "namespace": "default"
  },
  "spec": {
    "deregister": false,
    "deregistration": {
    },
    "entity_class": "proxy",
    "last_seen": 0,
    "sensu_agent_version": "",
    "subscriptions": null,
    "system": {
      "cloud_provider": "",
      "libc_type": "",
      "network": {
        "interfaces": null
```

```
      },
      "processes": null,
      "vm_role": "",
      "vm_system": ""
    }
  }
}
```

Service entities

***COMMERCIAL FEATURE**: Access business service monitoring (BSM), including service entities, in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

A service entity represents a business service in business service monitoring (BSM).Sensu processes service entity events just like events generated for agent and proxy entities.You can also use service entities for proxy check requests and events.

This example shows a service entity resource definition:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  name: postgresql
spec:
  entity_class: service
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "postgresql"
  },
  "spec": {
    "entity_class": "service"
  }
}
```

Backend entities

A backend entity represents a Sensu backend. Sensu automatically creates a backend entity for each backend when it is started and populates the entity with the backend's system information. Users cannot manually create backend entities.

Backends use their own entities to generate events due to error conditions like unavailable components and services.

This example shows a backend entity resource definition:

YML

```yaml
---
type: Entity
api_version: core/v2
metadata:
  name: 6b6264feda40
  namespace: sensu-system
spec:
  deregister: false
  deregistration: {}
  entity_class: backend
  last_seen: 0
  sensu_agent_version: ''
  subscriptions: null
  system:
    arch: amd64
    cloud_provider: ''
    hostname: 6b6264feda40
    libc_type: glibc
    network:
```

```
      interfaces:
        - addresses:
            - 127.0.0.1/8
          name: lo
        - addresses: null
          name: tunl0
        - addresses: null
          name: ip6tnl0
        - addresses:
            - 172.18.0.4/16
          mac: 02:42:ac:12:00:04
          name: eth0
    os: linux
    platform: redhat
    platform_family: rhel
    platform_version: '7.9'
    processes: null
    vm_role: guest
    vm_system: ''
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "6b6264feda40",
    "namespace": "sensu-system"
  },
  "spec": {
    "deregister": false,
    "deregistration": {
    },
    "entity_class": "backend",
    "last_seen": 0,
    "sensu_agent_version": "",
    "subscriptions": null,
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "6b6264feda40",
      "libc_type": "glibc",
```

```
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8"
            ],
            "name": "lo"
          },
          {
            "addresses": null,
            "name": "tunl0"
          },
          {
            "addresses": null,
            "name": "ip6tnl0"
          },
          {
            "addresses": [
              "172.18.0.4/16"
            ],
            "mac": "02:42:ac:12:00:04",
            "name": "eth0"
          }
        ]
      },
      "os": "linux",
      "platform": "redhat",
      "platform_family": "rhel",
      "platform_version": "7.9",
      "processes": null,
      "vm_role": "guest",
      "vm_system": ""
    }
  }
}
```


Usage limits

Sensu's usage limits are based on entities.

The free limit is 100 entities. All commercial features are available for free in the packaged Sensu Go distribution for up to 100 entities. If your Sensu instance includes more than 100 entities, contact us to learn how to upgrade your installation and increase your limit. Read the announcement on our blog for more information about our usage policy.

Commercial licenses may include an entity limit and entity class limits:

- Entity limit: the maximum number of entities of all classes your license includes. Agent, proxy, and service entities count toward the overall entity limit.
- Entity class limits: the maximum number of a specific class of entities (agent, proxy, or service) that your license includes.

For example, if your license has an entity limit of 10,000 and an agent entity class limit of 3,000, you cannot run more than 10,000 entities (agent and proxy) total. At the same time, you cannot run more than 3,000 agents. If you use only 1,500 agent entities, you can have 8,500 proxy entities before you reach the overall entity limit of 10,000.

If you have permission to create or update licenses, you will see messages in sensuctl and the web UI when you approach your licensed entity or entity class limit, as well as when you exceed these limits. You can also use sensuctl or the /license API to view your overall entity count and limit.

Entities reference

An entity represents anything that needs to be monitored, such as a server, container, or network switch, including the full range of infrastructure, runtime, and application types that compose a complete monitoring environment. Sensu uses agent entities, proxy entities, and service entities.

Sensu's free entity limit is 100 entities. All commercial features are available for free in the packaged Sensu Go distribution for up to 100 entities. If your Sensu instance includes more than 100 entities, contact us to learn how to upgrade your installation and increase your limit.

Learn more about entity limits in the license reference. Read the announcement on our blog for more information about our usage policy.

Create and manage agent entities

When an agent connects to a backend, the agent entity definition is created from the information in the `agent.yml` configuration file. The default `agent.yml` file location depends on your operating system

Agent entity example

This example shows the resource definition for an agent entity:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  name: webserver01
spec:
  deregister: false
  deregistration: {}
  entity_class: agent
  last_seen: 1542667231
  redact:
  - password
```

```
  - passwd
  - pass
  - api_key
  - api_token
  - access_key
  - secret_key
  - private_key
  - secret
subscriptions:
- entity:webserver01
system:
  arch: amd64
  libc_type: glibc
  vm_system: kvm
  vm_role: host
  cloud_provider: null
  processes:
  - name: Slack
    pid: 1349
    ppid: 0
    status: Ss
    background: true
    running: true
    created: 1582137786
    memory_percent: 1.09932518
    cpu_percent: 0.3263987595984941
  - name: Slack Helper
    pid: 1360
    ppid: 1349
    status: Ss
    background: true
    running: true
    created: 1582137786
    memory_percent: 0.146866455
    cpu_percent: 0.30897618146109257
    hostname: sensu2-centos
  network:
    interfaces:
    - addresses:
      - 127.0.0.1/8
      - ::1/128
      name: lo
```

```
      - addresses:
        - 10.0.2.15/24
        - fe80::26a5:54ec:cf0d:9704/64
        mac: 08:00:27:11:ad:d2
        name: enp0s3
      - addresses:
        - 172.28.128.3/24
        - fe80::a00:27ff:febc:be60/64
        mac: 08:00:27:bc:be:60
        name: enp0s8
    os: linux
    platform: centos
    platform_family: rhel
    platform_version: 7.4.1708
  sensu_agent_version: 1.0.0
  user: agent
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "webserver01"
  },
  "spec": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu2-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
```

```
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::26a5:54ec:cf0d:9704/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:bc:be:60",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:febc:be60/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
        "running": true,
```

```
            "created": 1582137786,
            "memory_percent": 0.146866455,
            "cpu_percent": 0.308976181461092553
          }
        ]
      },
      "sensu_agent_version": "1.0.0",
      "subscriptions": [
        "entity:webserver01"
      ],
      "last_seen": 1542667231,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ]
    }
  }
```


Manage agent entities via the backend

You can manage agent entities via the backend with sensuctl, the core/v2/entities API endpoints, and the web UI, just like any other Sensu resource. This means you do not need to update the `agent.yml` configuration file to add, update, or delete agent entity attributes like subscriptions and labels.

Management via the backend is the default configuration for agent entities.

***NOTE**: If you manage an agent entity via the backend, you cannot modify the agent entity with the `agent.yml` configuration file unless you delete the entity. In this case, the entity attributes in `agent.yml` are used only for initial entity creation unless you delete the entity.*

If you delete an agent entity that you modified with sensuctl, the core/v2/entities API endpoints, or the web UI, it will revert to the original configuration from `agent.yml`. If you change an agent entity's class to `proxy`, the backend will revert the change to `agent`.

Manage agent entities via the agent

If you prefer, you can manage agent entities via the agent rather than the backend. To do this, add the `agent-managed-entity` configuration option when you start the Sensu agent or set `agent-managed-entity: true` in your `agent.yml` file.

When you start an agent with the `agent-managed-entity` configuration option set to `true`, the agent becomes responsible for managing its entity configuration. An entity managed by this agent will include the label `sensu.io/managed_by: sensu-agent`. You cannot update these agent-managed entities via the Sensu backend REST API. To change an agent's configuration, restart the agent.

You can also maintain agent entities based on `agent.yml` by creating ephemeral agent entities with the deregister attribute set to `true`. With this setting, the agent entity will deregister every time the agent process stops and its keepalive expires. When it restarts, it will revert to the original configuration from `agent.yml`. You must set `deregister: true` in `agent.yml` before the agent entity is created.

Create and manage proxy entities

Proxy entities allow Sensu to monitor external resources on systems where you cannot install a Sensu agent, like a network switch or website.

You can create proxy entities the same way you would create agent entities, but Sensu can also dynamically create them when an entity does not already exist for a check result and add them to the entity store. In this case, Sensu will use the `proxy_entity_name` defined in the check to register proxy entities for your external resources.

Proxy entity registration differs from keepalive-based registration because the registration event happens while processing a check result instead of a keepalive message.

Modify proxy entities as needed via the backend with sensuctl, the core/v2/entities API endpoints, and the web UI.

> ***NOTE**: If you start an agent with the same name as an existing proxy entity, Sensu will change the proxy entity's class to `agent` and update its `system` field with information from the agent*

configuration.

Proxy entity example

This example shows the resource definition for a proxy entity:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  name: sensu-docs
spec:
  deregister: false
  deregistration: {}
  entity_class: proxy
  last_seen: 0
  sensu_agent_version: 1.0.0
  subscriptions: null
  system:
    cloud_provider: ""
    libc_type: ""
    network:
      interfaces: null
    processes: null
    vm_role: ""
    vm_system: ""
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-docs"
  },
  "spec": {
    "deregister": false,
    "deregistration": {
```

```
    },
    "entity_class": "proxy",
    "last_seen": 0,
    "sensu_agent_version": "1.0.0",
    "subscriptions": null,
    "system": {
      "cloud_provider": "",
      "libc_type": "",
      "network": {
        "interfaces": null
      },
      "processes": null,
      "vm_role": "",
      "vm_system": ""
    }
  }
}
```

Checks for proxy entities

Proxy entities allow Sensu to monitor external resources on systems or devices where a Sensu agent cannot be installed, like a network switch, website, or API endpoint.

You can configure a proxy check that includes a `proxy_entity_name` to associate the check results with a specific proxy entity. On the first check result, if the named proxy entity does not exist, Sensu will create it. You can also use proxy checks to monitor multiple proxy entities based on entity attributes specified in the check definition's `proxy_requests` attribute.

When you create a proxy check, make sure the check definition includes a subscription that matches the subscription of at least one agent entity to define which agents will run the check. Proxy entities do not use subscriptions.

Read Monitor external resources with proxy entities for details about creating proxy checks for one or more proxy entities.

Proxy entities and round robin scheduling

Proxy entities make round robin check scheduling more useful because they allow you to combine all round robin events into a single event. Instead of having a separate event for each agent entity, you

have a single event for the entire round robin.

If you don't use a proxy entity for round robin scheduling, you could have several failures in a row, but each event will only be aware of one of the failures.

If you use a proxy entity without round robin scheduling, and several agents share the same subscription, they will all execute the check for the proxy entity and you'll get duplicate results. When you enable round robin, you'll get one agent per interval executing the proxy check, but the event will always be listed under the proxy entity.

Use proxy entity filters to establish a many-to-many relationship between agent entities and proxy entities if you want even more power over the grouping.

Create and manage service entities

***COMMERCIAL FEATURE**: Access business service monitoring (BSM), including service entities, in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

Service entities are dynamically created entities that Sensu adds to the entity store when a service component generates an event. Service entities allow Sensu to monitor business services.

Create and modify service entities via the backend with sensuctl, the core/v2/entities API endpoints, and the web UI.

Service entity example

This example shows the resource definition for a service entity:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  name: postgresql
```

```
spec:
  entity_class: service
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "postgresql"
  },
  "spec": {
    "entity_class": "service"
  }
}
```

Backend entities

When a backend starts up, Sensu automatically creates the `sensu-system` namespace and a new entity with `entity_class: backend`. Sensu populates the backend entity with the backend's system information.

The backend uses its own entity to report cluster state errors. Backend entities can generate events due to error conditions like unavailable secrets providers or secrets. Events generated by a sensu-agent running on the backend host are also associated with the backend entity.

To prevent overloading the event bus with backend events, Sensu generates backend events no more than every 30 seconds, unless the status changes. Sensu does not generate keepalive events for backend entities.

Backend entity example

Here is an example definition for a backend entity:

YML

```
---
type: Entity
```

```yaml
api_version: core/v2
metadata:
  name: 6b6264feda40
  namespace: sensu-system
spec:
  deregister: false
  deregistration: {}
  entity_class: backend
  last_seen: 0
  sensu_agent_version: ''
  subscriptions: null
  system:
    arch: amd64
    cloud_provider: ''
    hostname: 6b6264feda40
    libc_type: glibc
    network:
      interfaces:
        - addresses:
            - 127.0.0.1/8
          name: lo
        - addresses: null
          name: tunl0
        - addresses: null
          name: ip6tnl0
        - addresses:
            - 172.18.0.4/16
          mac: 02:42:ac:12:00:04
          name: eth0
    os: linux
    platform: redhat
    platform_family: rhel
    platform_version: '7.9'
    processes: null
    vm_role: guest
    vm_system: ''
```

JSON

```json
{
  "type": "Entity",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "6b6264feda40",
    "namespace": "sensu-system"
  },
  "spec": {
    "deregister": false,
    "deregistration": {
    },
    "entity_class": "backend",
    "last_seen": 0,
    "sensu_agent_version": "",
    "subscriptions": null,
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "6b6264feda40",
      "libc_type": "glibc",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8"
            ],
            "name": "lo"
          },
          {
            "addresses": null,
            "name": "tunl0"
          },
          {
            "addresses": null,
            "name": "ip6tnl0"
          },
          {
            "addresses": [
              "172.18.0.4/16"
            ],
            "mac": "02:42:ac:12:00:04",
            "name": "eth0"
          }
        ]
      },
```

```
        "os": "linux",
        "platform": "redhat",
        "platform_family": "rhel",
        "platform_version": "7.9",
        "processes": null,
        "vm_role": "guest",
        "vm_system": ""
      }
    }
  }
```

Access backend entities

Only cluster admins have access to the `sensu-system` namespace and the backend entities it contains.

If you have cluster admin permissions, you can use sensuctl and the web UI to access backend entities like other entities.

Cluster admins who have write permissions for the `sensu-system` namespace can edit **only** labels and subscriptions for backend entities.

Backend entities and agent information

Sensu uses the same algorithm to determine backend entity names as the agent uses to determine entity names.

If a backend and an agent try to create the same entity, the entity class will default to `backend`. The information provided by the backend takes precedence over the information provided by the agent. The backend should update backend entities to use information from the backend instead of from the agent.

Delete a backend entity

Cluster admins can manually delete backend entities with sensuctl or the web UI.

Manage entity labels

Labels are custom attributes that Sensu includes with observation event data that you can use for response and web UI view searches. In contrast to annotations, you can use labels to filter API responses, sensuctl responses, and web UI search views.

Limit labels to metadata you need to use for response filtering and searches. For complex, non-identifying metadata that you will *not* need to use in response filtering and searches, use annotations rather than labels.

Agent entity labels

For new entities with class `agent`, you can define entity attributes in the `/etc/sensu/agent.yml` configuration file. For example, to add a `url` label, open `/etc/sensu/agent.yml` and add configuration for `labels`:

```
labels:
  url: sensu.docs.io
```

Or, use `sensu-agent start` configuration flags:

```
sensu-agent start --labels url=sensu.docs.io
```

***NOTE**: The entity attributes in `agent.yml` are used only for initial entity creation. Modify existing agent entities via the backend with sensuctl, the core/v2/entities API endpoints, and the web UI.*

Proxy entity labels

For entities with class `proxy`, you can create and manage labels with sensuctl.

For example, suppose you have a proxy entity like this one:

YML

```
---
type: Entity
api_version: core/v2
```

```
metadata:
  labels:
    url: docs.sensu.io
  name: sensu-docs
spec:
  deregister: false
  entity_class: proxy
  sensu_agent_version: 1.0.0
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "labels": {
      "url": "docs.sensu.io"
    },
    "name": "sensu-docs"
  },
  "spec": {
    "deregister": false,
    "entity_class": "proxy",
    "sensu_agent_version": "1.0.0"
  }
}
```

To add a `proxy_type` label to this existing entity, run the following command to open the entity definition:

```
sensuctl edit entity sensu-docs
```

Then, update the metadata scope in the entity definition to add the `proxy_type` label as shown below:

YML

```
---
```

```
type: Entity
api_version: core/v2
metadata:
  labels:
    url: docs.sensu.io
    proxy_type: website
  name: sensu-docs
spec:
  '...': '...'
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "labels": {
      "url": "docs.sensu.io",
      "proxy_type": "website"
    },
    "name": "sensu-docs"
  },
  "spec": {
    "...": "..."
  }
}
```

Save your changes to update the proxy entity definition with the `proxy_type` label.

Service entity labels

For entities with class `service`, you can create and manage labels with sensuctl. To create a service entity with a `service_type` label using sensuctl `create`, create a file called `service-entity.json` with an entity definition that includes `labels`:

YML

```
---
type: Entity
```

```
api_version: core/v2
metadata:
  name: postgresql
  labels:
    service_type: datastore
spec:
  entity_class: service
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "postgresql",
    "labels": {
      "service_type": "datastore"
    }
  },
  "spec": {
    "entity_class": "service"
  }
}
```

Then run sensuctl create to create the entity based on the definition:

SHELL

```
sensuctl create --file service-entity.yml
```

SHELL

```
sensuctl create --file service-entity.json
```

To add a label to an existing service entity, use sensuctl edit. For example, to add a `region` label to a `postgresql` entity:

```
```

```
sensuctl edit entity postgresql
```

And update the metadata scope to include the `region` label:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  labels:
    service_type: datastore
    region: us-west-1
  name: postgresql
spec:
  '...': '...'
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "labels": {
      "service_type": "datastore",
      "region": "us-west-1"
    },
    "name": "postgresql"
  },
  "spec": {
    "...": "..."
  }
}
```

Entities specification

Top-level attributes

api_version

description	Top-level attribute that specifies the Sensu API group and version. For entities in this version of Sensu, this attribute should always be `core/v2`.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

example

```
api_version: core/v2
```

JSON

```
{
  "api_version": "core/v2"
}
```

metadata

description	Top-level collection of metadata about the entity, including `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the entity definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

example

```
metadata:
  name: webserver01
  namespace: default
```

```
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    slack-channel: "#monitoring"
```

JSON

```
{
  "metadata": {
    "name": "webserver01",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "slack-channel": "#monitoring"
    }
  }
}
```

spec	
description	Top-level map that includes the entity spec attributes.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  entity_class: agent` `  system:` `  hostname: sensu2-centos` `  os: linux` `  platform: centos`

```
platform_family: rhel
platform_version: 7.4.1708
network:
  interfaces:
  - name: lo
    addresses:
    - 127.0.0.1/8
    - "::1/128"
  - name: enp0s3
    mac: '08:00:27:11:ad:d2'
    addresses:
    - 10.0.2.15/24
    - fe80::26a5:54ec:cf0d:9704/64
  - name: enp0s8
    mac: '08:00:27:bc:be:60'
    addresses:
    - 172.28.128.3/24
    - fe80::a00:27ff:febc:be60/64
arch: amd64
libc_type: glibc
vm_system: kvm
vm_role: host
cloud_provider: ''
processes:
- name: Slack
  pid: 1349
  ppid: 0
  status: Ss
  background: true
  running: true
  created: 1582137786
  memory_percent: 1.09932518
  cpu_percent: 0.3263987595984941
- name: Slack Helper
  pid: 1360
  ppid: 1349
  status: Ss
  background: true
  running: true
  created: 1582137786
  memory_percent: 0.146866455
  cpu_percent: 0.30897618146109257
```

```
  sensu_agent_version: 1.0.0
  subscriptions:
  - entity:webserver01
  last_seen: 1542667231
  deregister: false
  deregistration: {}
  user: agent
  redact:
  - password
  - passwd
  - pass
  - api_key
  - api_token
  - access_key
  - secret_key
  - private_key
  - secret
```

JSON

```
{
  "spec": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu2-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
```

```
            "10.0.2.15/24",
            "fe80::26a5:54ec:cf0d:9704/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:bc:be:60",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:febc:be60/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 0.146866455,
        "cpu_percent": 0.30897618146109257
      }
```

```
        ]
      },
      "sensu_agent_version": "1.0.0",
      "subscriptions": [
        "entity:webserver01"
      ],
      "last_seen": 1542667231,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ]
    }
}
```

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Entities should always be type `Entity`.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: Entity` **JSON**

```
{
  "type": "Entity"
}
```

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views. ***NOTE**: For annotations defined in agent.yml or backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the annotation `webhookURL: "https://my-webhook.com"` in agent.yml or backend.yml, it will be listed as `webhookurl: "https://my-webhook.com"` in entity definitions.* *Key cases are **not** modified for annotations you define with a command line flag or an environment variable.*
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` `  playbook: www.example.url` **JSON**

```
{
  "annotations": {
    "managed-by": "ops",
    "playbook": "www.example.url"
  }
}
```

created_by

description	Username of the Sensu user who created the entity or last updated the entity. Sensu automatically populates the `created_by` field when the entity is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.

Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.

> ***NOTE**: For labels that you define in agent.yml or backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the label `proxyType: "website"` in agent.yml or backend.yml, it will be listed as `proxytype: "website"` in entity definitions.*
>
> *Key cases are **not** modified for labels you define with a command line flag or an environment variable.*

required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

name

description	Unique name of the entity, validated with Go regex `\A[\w\.\-]+\z` .
required	true
type	String
example	**YML** `name: example-hostname` **JSON** `{ "name": "example-hostname" }`

namespace

description	Sensu RBAC namespace that this entity belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: production` **JSON** `{ "namespace": "production" }`

Spec attributes

deregister	
description	If the entity should be removed when it stops sending keepalive messages, `true`. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	**YML** `deregister: false` **JSON** `{ "deregister": false }`

deregistration	
description	Map that contains a handler name to use when an agent entity is deregistered. Read deregistration attributes for more information.
required	false
type	Map
example	**YML** `deregistration:` `  handler: email-handler` **JSON**

```
{
  "deregistration": {
    "handler": "email-handler"
  }
}
```

entity_class

description	Entity type, validated with Go regex `\A[\w\.\-]+\z`. Class names have special meaning. An entity that runs an agent is class `agent` and is reserved. Setting the value of `entity_class` to `proxy` creates a proxy entity. An entity that represents a business service is class `service`. For other types of entities, the `entity_class` attribute isn't required, and you can use it to indicate an arbitrary type of entity (like `lambda` or `switch`).
required	true
type	String

YML

example

```
entity_class: agent
```

JSON

```
{
  "entity_class": "agent"
}
```

last_seen

description	Time at which the entity was last seen. In seconds since the Unix epoch.
required	false

type	Integer

example

YML

```
last_seen: 1522798317
```

JSON

```
{
  "last_seen": 1522798317
}
```

redact

description	List of items to redact from log messages. If a value is provided, it overwrites the default list of items to be redacted.
required	false
type	Array
default	[“password”, “passwd”, “pass”, “api_key”, “api_token”, “access_key”, “secret_key”, “private_key”, “secret”]

example

YML

```
redact:
- extra_secret_tokens
```

JSON

```
{
  "redact": [
    "extra_secret_tokens"
  ]
}
```

sensu_agent_version

description	Sensu Semantic Versioning (SemVer) version of the agent entity.
required	true
type	String
example	**YML** `sensu_agent_version: 1.0.0` **JSON** `{ "sensu_agent_version": "1.0.0" }`

subscriptions

description	List of subscription names for the entity. The entity by default has an entity-specific subscription, in the format of `entity:<name>` where `name` is the entity's hostname.
required	false
type	Array
default	The entity-specific subscription.
example	**YML** `subscriptions:` `- web` `- prod` `- entity:example-entity` **JSON** `{`

```
  "subscriptions": [
    "web",
    "prod",
    "entity:example-entity"
  ]
}
```

system

description	System information about the entity, such as operating system and platform. Read system attributes for more information. ***NOTE**: Process discovery is disabled in this version of Sensu. New events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.*
required	false
type	Map **YML**
example	

```yaml
system:
  arch: amd64
  libc_type: glibc
  vm_system: kvm
  vm_role: host
  cloud_provider: null
  processes:
  - name: Slack
    pid: 1349
    ppid: 0
    status: Ss
    background: true
    running: true
    created: 1582137786
    memory_percent: 1.09932518
    cpu_percent: 0.3263987595984941
```

```
  - name: Slack Helper
    pid: 1360
    ppid: 1349
    status: Ss
    background: true
    running: true
    created: 1582137786
    memory_percent: 0.146866455
    cpu_percent: 0.30897618146109257
  hostname: example-hostname
  network:
    interfaces:
    - addresses:
      - 127.0.0.1/8
      - ::1/128
      name: lo
    - addresses:
      - 93.184.216.34/24
      - 2606:2800:220:1:248:1893:25c8:1946/10
      mac: 52:54:00:20:1b:3c
      name: eth0
  os: linux
  platform: ubuntu
  platform_family: debian
  platform_version: "16.04"
```

JSON

```
{
  "system": {
    "hostname": "example-hostname",
    "os": "linux",
    "platform": "ubuntu",
    "platform_family": "debian",
    "platform_version": "16.04",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
```

```
          ]
        },
        {
          "name": "eth0",
          "mac": "52:54:00:20:1b:3c",
          "addresses": [
            "93.184.216.34/24",
            "2606:2800:220:1:248:1893:25c8:1946/10"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 0.146866455,
        "cpu_percent": 0.308976181461092553
      }
    ]
  }
```

```
}
```

user	
description	Sensu RBAC username used by the entity. Agent entities require get, list, create, update, and delete permissions for events across all namespaces.
type	String
default	`agent`
example	**YML** ``` user: agent ``` **JSON** ``` { "user": "agent" } ```

Deregistration attributes

handler	
description	Name of the handler to call when an agent entity is deregistered.
required	false
type	String
example	**YML** ``` handler: email-handler ```

JSON

```
{
  "handler": "email-handler"
}
```

System attributes

arch

description	Entity's system architecture. This value is determined by the Go binary architecture as a function of runtime.GOARCH. An `amd` system running a `386` binary will report the `arch` as `386`.
required	false
type	String

YML

example

```
arch: amd64
```

JSON

```
{
  "arch": "amd64"
}
```

arm_version

description	Entity's ARM version. Automatically populated upon agent startup for entities with ARM system architecture. For entities that do not use ARM system architecture, the `arm_version` attribute is omitted from the entity definition.

required	false
type	Integer

example

YML

```yaml
arm_version: 7
```

JSON

```json
{
  "arm_version": 7
}
```

cloud_provider

description

Entity's cloud provider environment. Automatically populated upon agent startup if the `detect-cloud-provider` configuration option is set. Returned empty unless the agent runs on Amazon Elastic Compute Cloud (EC2), Google Cloud Platform (GCP), or Microsoft Azure.

> ***NOTE**: This feature can result in several HTTP requests or DNS lookups being performed, so it may not be appropriate for all environments.*

required	false
type	String

example

YML

```yaml
"cloud_provider": ""
```

JSON

```json
{
  "cloud_provider": ""
}
```

float_type

description	Type of float the entity's system architecture uses: `hardfloat` or `softfloat`. Automatically populated upon agent startup for entities with MIPS, MIPS LE, MIPS 64, or MIPS 64 LE system architecture. For entities that do not use a MIPS system architecture, the `float_type` attribute is omitted from the entity definition.
required	false
type	String

example

YML

```
float_type: hardfloat
```

JSON

```
{
  "float_type": "hardfloat"
}
```

hostname

description	Hostname of the entity.
required	false
type	String

example

YML

```
hostname: example-hostname
```

JSON

```
```

```
{
  "hostname": "example-hostname"
}
```

libc_type

description	Entity's libc type.
required	false
type	String
example	**YML** `libc_type: glibc` **JSON** `{ "libc_type": "glibc" }`

network

description	Entity's network interface list. Read network attributes for more information.
required	false
type	Map
example	**YML** `network:` `  interfaces:` `  - addresses:` `  - 127.0.0.1/8`

```
    - ::1/128
    name: lo
  - addresses:
    - 93.184.216.34/24
    - 2606:2800:220:1:248:1893:25c8:1946/10
    mac: 52:54:00:20:1b:3c
    name: eth0
```

JSON

```
{
  "network": {
    "interfaces": [
      {
        "name": "lo",
        "addresses": [
          "127.0.0.1/8",
          "::1/128"
        ]
      },
      {
        "name": "eth0",
        "mac": "52:54:00:20:1b:3c",
        "addresses": [
          "93.184.216.34/24",
          "2606:2800:220:1:248:1893:25c8:1946/10"
        ]
      }
    ]
  }
}
```

os	
description	Entity’s operating system.
required	false

type	String

YML

example

```yaml
os: linux
```

JSON

```json
{
  "os": "linux"
}
```

platform

description	Entity's operating system distribution.
required	false
type	String

YML

example

```yaml
platform: ubuntu
```

JSON

```json
{
  "platform": "ubuntu"
}
```

platform_family

description	Entity's operating system family.
required	false

type	String

YML

example

```
platform_family: debian
```

JSON

```
{
  "platform_family": "debian"
}
```

platform_version

description	Entity's operating system version.
required	false
type	String

YML

example

```
platform_version: 16.04
```

JSON

```
{
  "platform_version": "16.04"
}
```

processes

description

List of processes on the local agent. Read processes attributes for more information.

***NOTE**: Process discovery is disabled in this version of Sensu. New*

events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.

required	false
type	Map
example	**YML** `processes:` `- name: Slack` `  pid: 1349` `  ppid: 0` `  status: Ss` `  background: true` `  running: true` `  created: 1582137786` `  memory_percent: 1.09932518` `  cpu_percent: 0.3263987595984941` `- name: Slack Helper` `  pid: 1360` `  ppid: 1349` `  status: Ss` `  background: true` `  running: true` `  created: 1582137786` `  memory_percent: 0.146866455` `  cpu_percent: 0.30897618146109257` **JSON** `{` `  "processes": [` `  {` `  "name": "Slack",` `  "pid": 1349,` `  "ppid": 0,` `  "status": "Ss",` `  "background": true,` `  "running": true,` `  "created": 1582137786,`

```
      "memory_percent": 1.09932518,
      "cpu_percent": 0.3263987595984941
    },
    {
      "name": "Slack Helper",
      "pid": 1360,
      "ppid": 1349,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 0.146866455,
      "cpu_percent": 0.308976181461092553
    }
  ]
}
```


vm_role

description	Entity's virtual machine role. Automatically populated upon agent startup.
required	false
type	String
example	**YML** `vm_role: host` **JSON** `{ "vm_role": "host" }`

vm_system	
description	Entity’s virtual machine system. Automatically populated upon agent startup.
required	false
type	String
example	**YML** `vm_system: kvm` **JSON** `{ "vm_system": "kvm" }`

Network attributes

interfaces	
description	List of network interfaces available on the entity, with their associated MAC and IP addresses. Read interfaces attributes for more information.
required	false
type	Array
example	**YML** `interfaces:` `- addresses:` `  - 127.0.0.1/8` `  - ::1/128` `  name: lo` `- addresses:` `  - 93.184.216.34/24` `  - 2606:2800:220:1:248:1893:25c8:1946/10`

```
  mac: 52:54:00:20:1b:3c
  name: eth0
```

JSON

```
{
  "interfaces": [
    {
      "name": "lo",
      "addresses": [
        "127.0.0.1/8",
        "::1/128"
      ]
    },
    {
      "name": "eth0",
      "mac": "52:54:00:20:1b:3c",
      "addresses": [
        "93.184.216.34/24",
        "2606:2800:220:1:248:1893:25c8:1946/10"
      ]
    }
  ]
}
```

Interfaces attributes

addresses	
description	List of IP addresses for the network interface.
required	false
type	Array
example	**YML** `addresses:`

```
- 93.184.216.34/24
- 2606:2800:220:1:248:1893:25c8:1946/10
```

JSON

```
{
  "addresses": [
    "93.184.216.34/24",
    "2606:2800:220:1:248:1893:25c8:1946/10"
  ]
}
```

mac

description	Network interface's MAC address.
required	false
type	string
example	**YML** `mac: 52:54:00:20:1b:3c` **JSON** `{ "mac": "52:54:00:20:1b:3c" }`

name

description	Network interface name.
required	false

type	String

YML

example	`name: eth0`

JSON

```
{
  "name": "eth0"
}
```

Processes attributes

***COMMERCIAL FEATURE**: Access processes attributes with the `discover-processes` configuration option in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Process discovery is disabled in this version of Sensu. New events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.*

background

description	If `true`, the process is a background process. Otherwise, `false`.
required	false
type	Boolean

YML

example	`background: true`

JSON

```
{
```

```
  "background": true
}
```

cpu_percent

description	Percent of CPU the process is using. The value is returned as a floating-point number where 0.0 = 0% and 1.0 = 100%. For example, the cpu_percent value 0.12639 equals 12.639%. ***NOTE**: The `cpu_percent` attribute is supported on Linux and macOS.It is not supported on Windows.*
required	false
type	float
example	**YML** `cpu_percent: 0.12639` **JSON** `{ "cpu_percent": 0.12639 }`

created

description	Time at which the process was created. In seconds since the Unix epoch.
required	false
type	Integer

YML

example

```yaml
created: 1586138786
```

JSON

```json
{
  "created": 1586138786
}
```

memory_percent

description

Percent of memory the process is using. The value is returned as a floating-point number where 0.0 = 0% and 1.0 = 100%. For example, the memory_percent value 0.19932 equals 19.932%.

> ***NOTE**: The `memory_percent` attribute is supported on Linux and macOS.It is not supported on Windows.*

required

false

type

float

YML

example

```yaml
memory_percent: 0.19932
```

JSON

```json
{
  "memory_percent": 0.19932
}
```

name

description	Name of the process.
required	false
type	String
example	**YML** `name: Slack` **JSON** `{ "name": "Slack" }`

pid

description	Process ID of the process.
required	false
type	Integer
example	**YML** `pid: 1349` **JSON** `{ "pid": 1349 }`

ppid

description	Parent process ID of the process.
required	false
type	Integer
example	**YML** `ppid: 0` **JSON** `{ "ppid": 0 }`

running

description	If `true`, the process is running. Otherwise, `false`.
required	false
type	Boolean
example	**YML** `running: true` **JSON** `{ "running": true }`

status	
description	Status of the process. Read the Linux `top` manual page for examples.
required	false
type	String
example	**YML** `status: Ss` **JSON** `{ "status": "Ss" }`

Automatically register and deregister entities

Sensu uses the publish/subscribe pattern of communication, which allows automated registration and deregistration of ephemeral systems. Sensu agents automatically discover and register infrastructure components and the services running on them. At the same time, when an agent process stops, the Sensu backend can automatically create and process a deregistration event.

Automatic registration and deregistration keeps your Sensu instance up-to-date and avoids unnecessary process load, especially in containerized environments where containers routinely come online and offline. You'll see observability event data soon after an agent entity comes online, and you won't receive stale events or alerts for entities that no longer exist.

You can also configure handlers that take specific actions based on agent registration and deregistration, such as updating external configuration management databases (CMDBs).

Discovery and registration

Sensu agents automatically discover and register infrastructure components and the services running on them.

> ***NOTE**: Automatic discovery is not supported for proxy entities because they cannot run a Sensu agent. Use the core/v2/events API to send manual keepalive events for proxy entities.*

Registration events

When an agent comes online, it sends its first keepalive event. When a Sensu backend processes a keepalive event for an agent whose name is not already listed in the Sensu agent registry, Sensu automatically registers the agent. The Sensu backend stores the entity registry, which you can view by running `sensuctl entity list`.

If you configure a handler named `registration`, the Sensu backend will create and process a registration event for that handler to process. The `registration` handler must reference the name of a handler or handler set that you want to execute for every registration event.

***WARNING**: Registration events are not stored in the event registry, so they are not accessible via the Sensu API. However, all registration events are logged in the Sensu backend log.*

Registration handler example

You can use registration event handlers to execute one-time handlers for new Sensu agents based on registration events.

For example, suppose you want to update the ServiceNow CMDB table that contains your Sensu entity inventory upon every registration event. First, configure a handler that uses the sensu/sensu-servicenow-handler dynamic runtime asset and the `--cmdb-registration` argument:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: servicenow_cmdb
spec:
  type: pipe
  command: sensu-servicenow-handler --cmdb-registration
  runtime_assets:
  - sensu/sensu-servicenow-handler:3.0.0
  env_vars:
  - SERVICENOW_URL=https://example.servicenow.com
  secrets:
  - name: SERVICENOW_USERNAME
    secret: servicenow_username
  - name: SERVICENOW_PASSWORD
    secret: servicenow_password
  timeout: 10
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "servicenow_cmdb"
  },
  "spec": {
    "type": "pipe",
    "command": "sensu-servicenow-handler --cmdb-registration",
    "runtime_assets": [
      "sensu/sensu-servicenow-handler:3.0.0"
    ],
    "env_vars": [
      "SERVICENOW_URL=https://example.servicenow.com"
    ],
    "secrets": [
      {
        "name": "SERVICENOW_USERNAME",
        "secret": "servicenow_username"
      },
      {
        "name": "SERVICENOW_PASSWORD",
        "secret": "servicenow_password"
      }
    ],
    "timeout": 10
  }
}
```

Then, create a `registration` handler that references the `servicenow_cmdb` handler:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: registration
spec:
  handlers:
  - servicenow_cmdb
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "registration"
  },
  "spec": {
    "handlers": [
      "servicenow_cmdb"
    ],
    "type": "pipe"
  }
}
```

Now the Sensu backend will execute the referenced `servicenow-cmdb` handler for every registration event. The *referenced* handler can send registration event alerts to any service, such as Sumo Logic or PagerDuty, as long as it is listed within a handler named `registration`.

***PRO TIP**: Use a handler set to execute multiple handlers in response to registration events.*

Deregistration

Just like Sensu can automatically register new agent entities when they send their first keepalive, Sensu can automatically deregister agent entities when they shut down and the agent process stops.

To enable automatic deregistration, set the agent `deregister` attribute to `true`. When the Sensu agent process stops and the agent stops sending keepalive messages, the Sensu backend can deregister the corresponding entity without any further action.

***NOTE**: Deregistration is supported for agent entities that have sent at least one keepalive. Deregistration is **not** supported for proxy entities, which do not send keepalives, and the backend does not automatically create and process deregistration events for proxy entities.*

Deregistration events

As with registration events, the Sensu backend can create and process a deregistration event when a Sensu agent process stops.

When an agent exceeds its keepalive timeout setting, the backends will generate a keepalive failure for that agent and create an event on its behalf. If you set the agent `deregister` attribute to `true`, when keepalive failure occurs, Sensu will delete the agent entity from the entity registry and send a deregistration event through the event pipeline.

To take action based on deregistration events, you must also specify a handler to use for deregistration events in the agent or backend configuration:

- To use a deregistration handler for a specific agent, set the **agent** deregistration-handler attribute.
- To use a deregistration handler to process all deregistration events for all agents, set the **backend** deregistration-handler attribute.

The agent `deregistration-handler` attribute overrides the backend `deregistration-handler` attribute. In other words, if you specify both an agent and backend deregistration handler, Sensu will use only the handler specified in the agent configuration.

> ***NOTE**: If you set the agent `deregister` attribute to `true`, when a Sensu agent process stops, the Sensu backend will deregister the corresponding entity.*
>
> *Deregistration prevents and clears alerts for failing keepalives for agent entities — the backend does not distinguish between intentional shutdown and failure. As a result, if you set the deregister flag to `true` and an agent process stops for any reason, you will not receive alerts for keepalive events in the web UI.*
>
> *If you want to receive alerts for failing keepalives, set the agent `deregister` attribute to `false`.*

Deregistration handler example

Just like registration events, deregistration events can trigger a one-time handler that performs an action like updating an external CMDB or ephemeral infrastructures. In fact, you can use the `servicenow_cmdb` handler to update the ServiceNow CMDB table that contains your Sensu entity inventory, this time based on every deregistration event.

To specify `servicenow_cmdb` as the agent deregistration handler:

SHELL

```
sensu-agent start --deregistration-handler servicenow_cmdb
```

SHELL

```
deregistration-handler: servicenow_cmdb
```

Next steps

The Sensu Catalog includes the Platform Discovery integration, which detects the agent operating system and platform information and updates the agent's subscriptions accordingly. This integration allows you to deploy agents with a single subscription and use the auto-discovery check to add system-based subscriptions automatically.

Follow Create limited service accounts to automatically remove AWS EC2 instances that are not in a pending or running state.

Monitor external resources with proxy entities

Proxy entities allow Sensu to monitor external resources on systems and devices where a Sensu agent cannot be installed, like a network switch or a website. You can create proxy entities with sensuctl, the Sensu API, and the `proxy_entity_name` check attribute. When executing checks that include a `proxy_entity_name` or `proxy_requests` attribute, Sensu agents report the resulting event under the proxy entity instead of the agent entity.

This guide explains how to use a proxy entity to monitor website status and includes two methods for configuring the required Sensu resources:

- Follow the Sensu Catalog integration method to configure the resources you need directly in Sensu web UI.
- Follow the command line configuration method to manually create the Sensu resources you need.

This guide also explains how to use proxy checks to monitor a group of websites, with command line configuration instructions.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

Use a proxy entity to monitor a website (Sensu Catalog configuration)

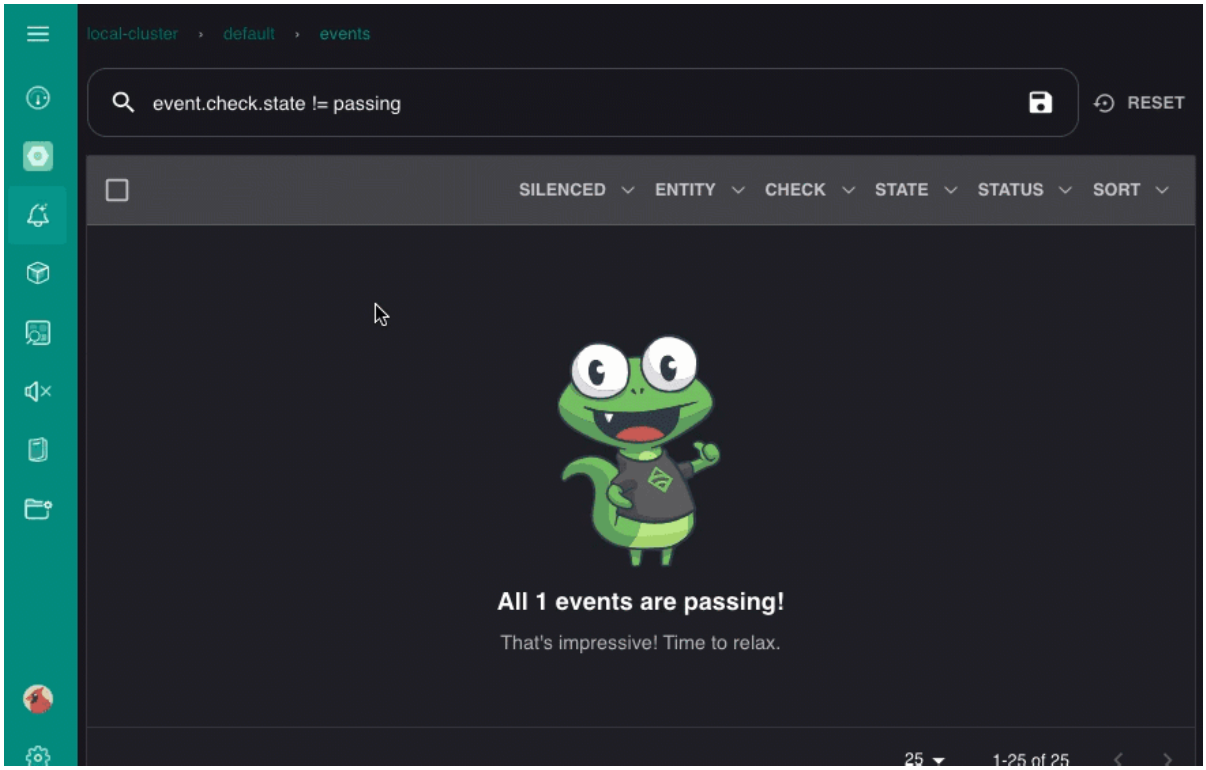
Follow the steps in this section to use the Sensu Catalog to configure status monitoring for sensu.io. You'll configure a check with a proxy entity name and Sensu will create an entity to represent sensu.io and report the status of the site under this entity.

The Sensu Catalog is part of the Sensu web UI, so you can complete all the necessary configuration directly from your browser.

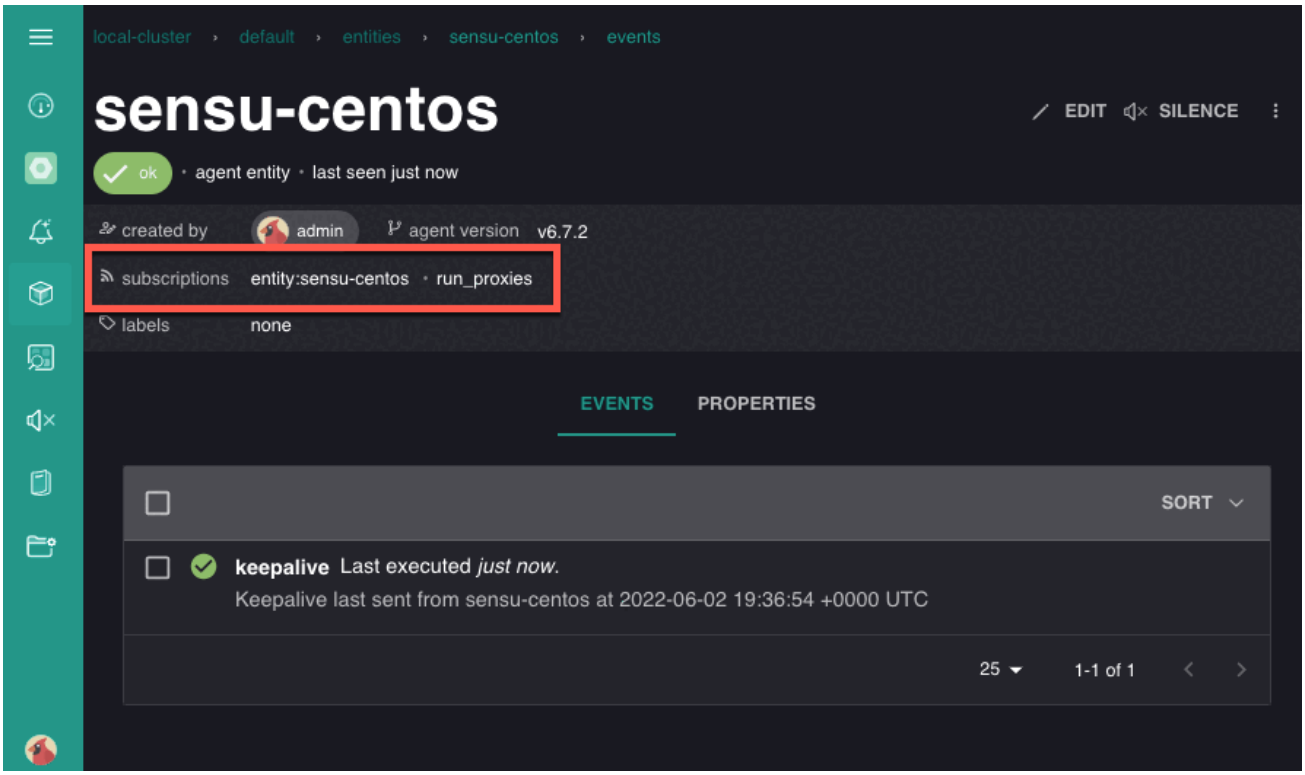
Configure a Sensu entity

To run the proxy entity check, you'll need a Sensu agent entity with the subscription `run_proxies`. Here's how to add the subscription:

1. In the web UI, navigate to the Entities page.
2. Click the agent entity you want to use to run your check.
3. At the top right corner of the individual entity's page, click **EDIT** to open the Edit Entity dialog.
4. Under **Schedule**, type `run_proxies` in the **Subscriptions** and press **Return**.
5. Click **SUBMIT** to save your changes.



On the individual entity's page, the **subscriptions** should now include `run_proxies`.

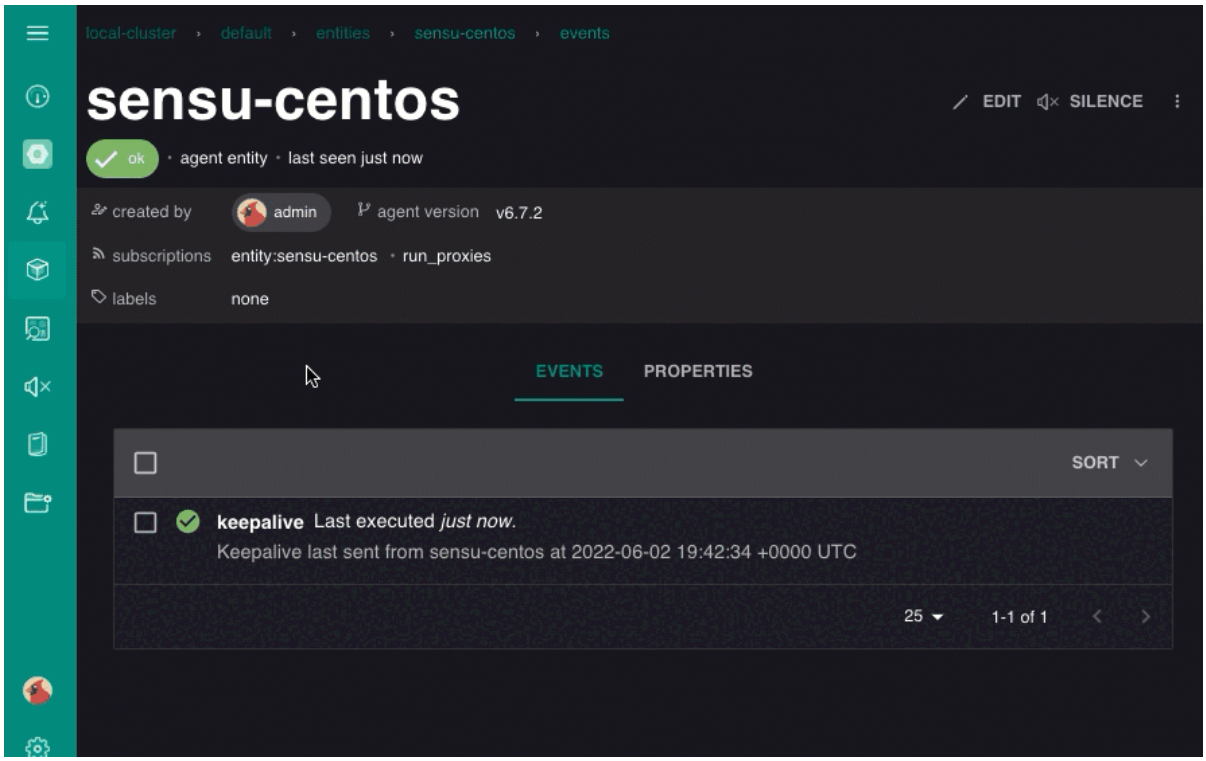


Create the check with a Sensu Catalog integration

With your entity subscription configured, you can use the Sensu Catalog to create the check you need to monitor sensu.io.

1. In the web UI, navigate to the Sensu Catalog page.

2. In the catalog menu on the left, click **Service monitoring** and click the **HTTP Endpoint Monitoring (Remote)** integration.



3. At the top right corner of the page, click **INSTALL…** to open the HTTP Endpoint Configuration dialog page.

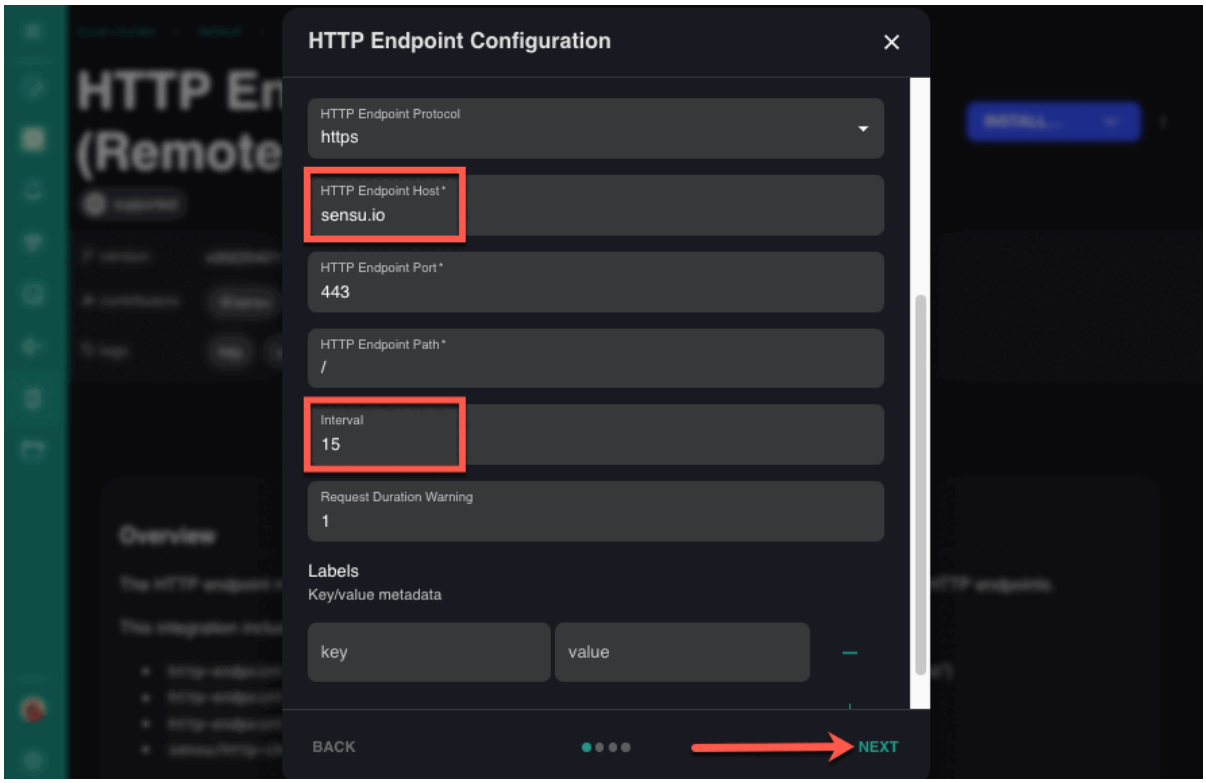
   Installing the HTTP Endpoint Monitoring (Remote) integration will add the following resources to your Sensu instance:

   - The `sensu/http-checks` asset
   - Two checks: one to produce endpoint status events and one to collect endpoint metrics
   - A new proxy entity to represent sensu.io

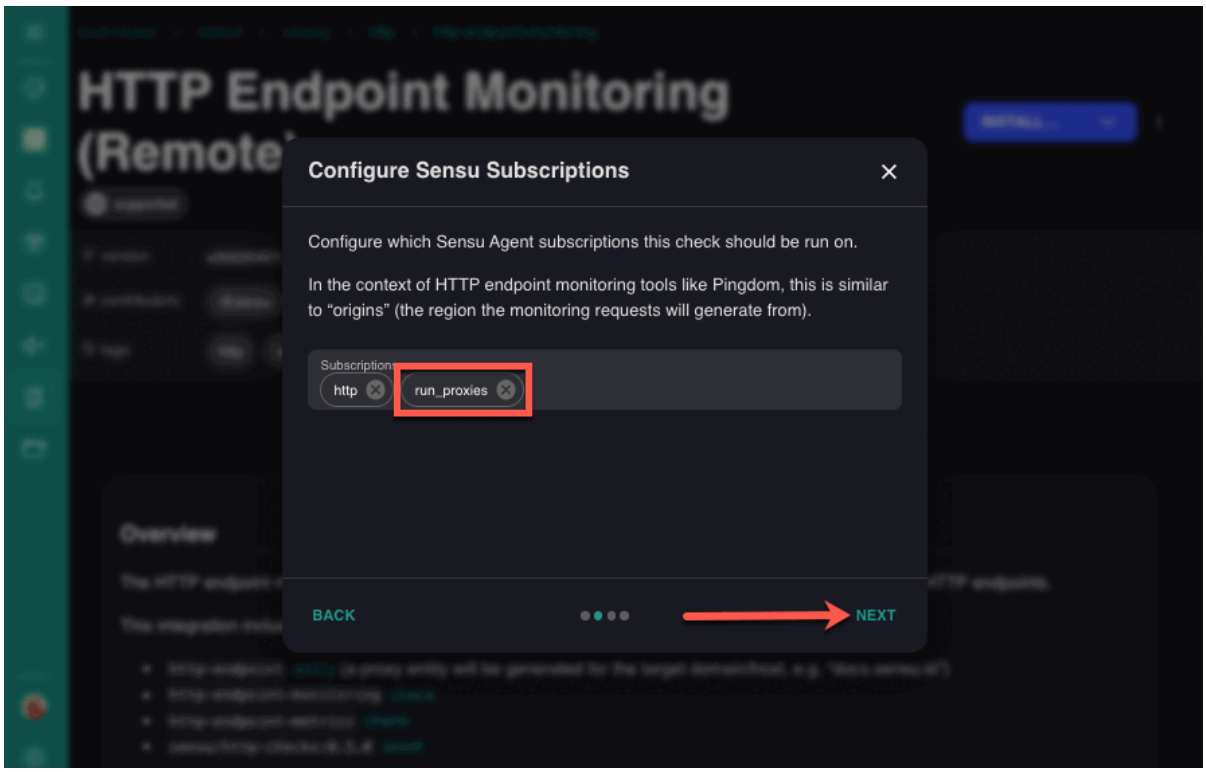
4. In the HTTP Endpoint Configuration dialog page, update the values in the `HTTP Endpoint Host` and `Interval` fields:

   - HTTP Endpoint Host: type `sensu.io`
   - Interval: type `15`

   After you update the values, click **NEXT**.

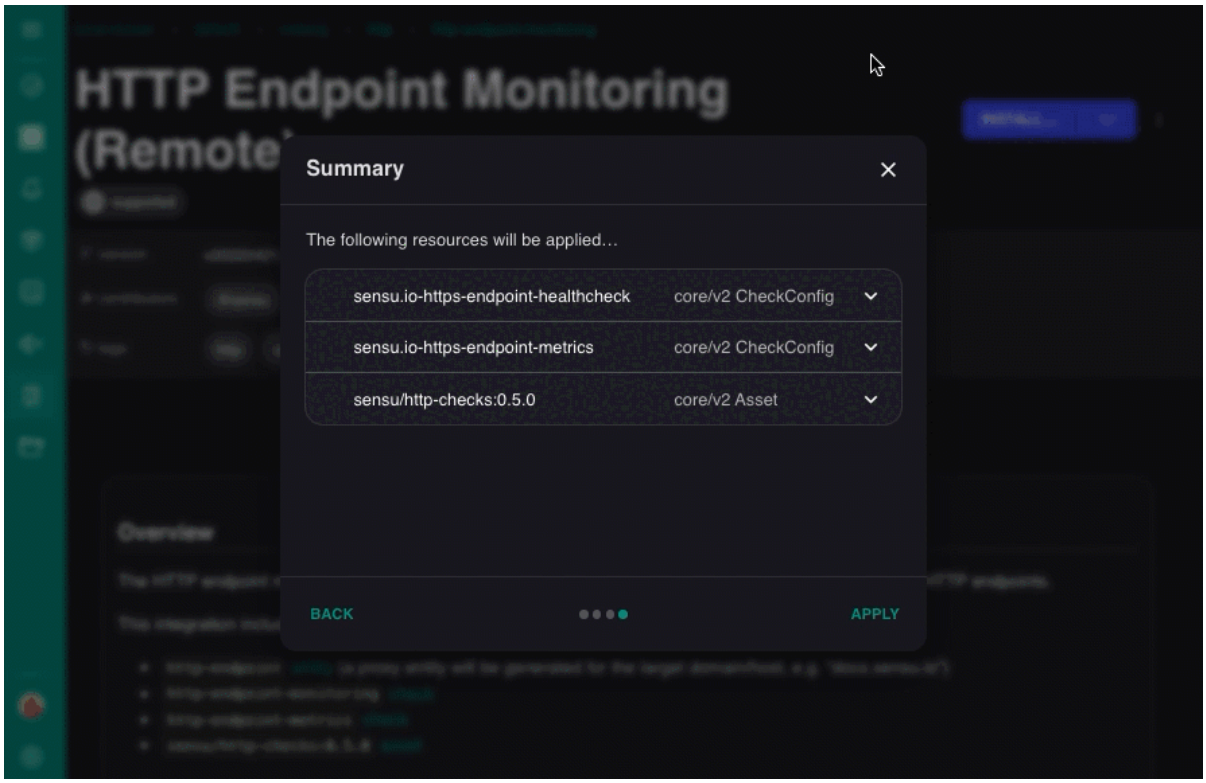


5. In the Configure Sensu Subscriptions dialog page, type `run_proxies` in the Subscriptions field and press **Return**.After you add the subscription, click **NEXT**.



6. The HTTP Endpoint Monitoring (Remote) integration in the Sensu Catalog includes a dialog page for adding pipelines to filter and handle your check’s events.If you already have a pipeline to use, you can add it now.Otherwise, click **NEXT** to skip this step.

7. The Summary dialog page lists definitions for the resources that the integration will add.Click the down-arrow next to any resource to view its complete definition in YAML or JSON format.



8. Click **APPLY** to save the asset and check definitions for the integration.

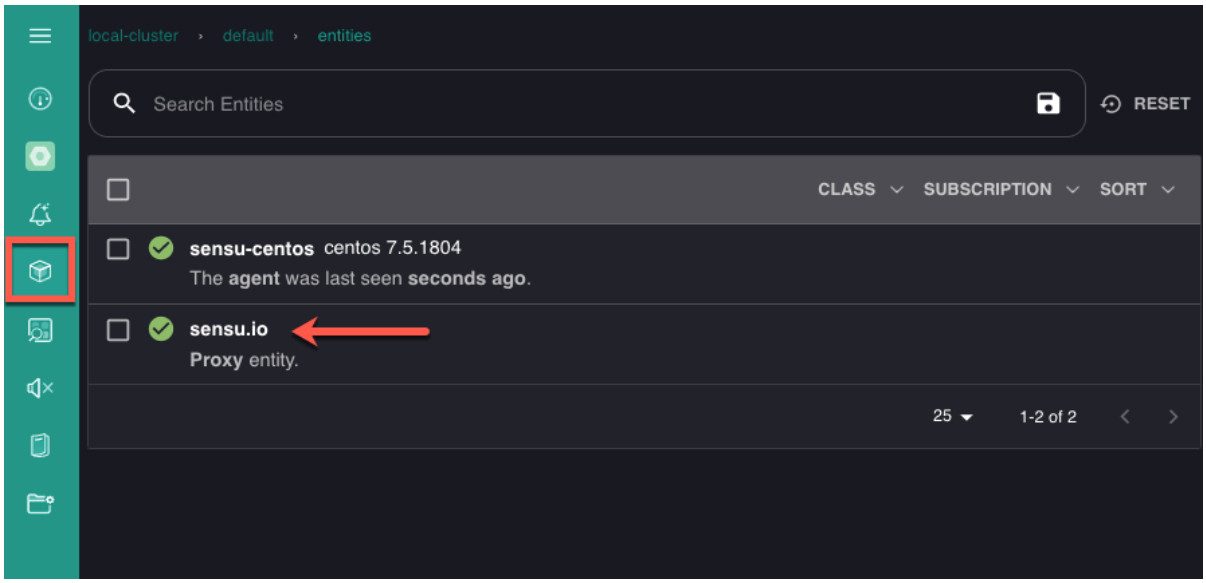
9. Click **FINISH** to return to the integration page.

Validate the check

To make sure that the monitoring check is working properly, confirm that Sensu created an entity to represent sensu.io and the `http-endpoint-healthcheck` check is producing events.

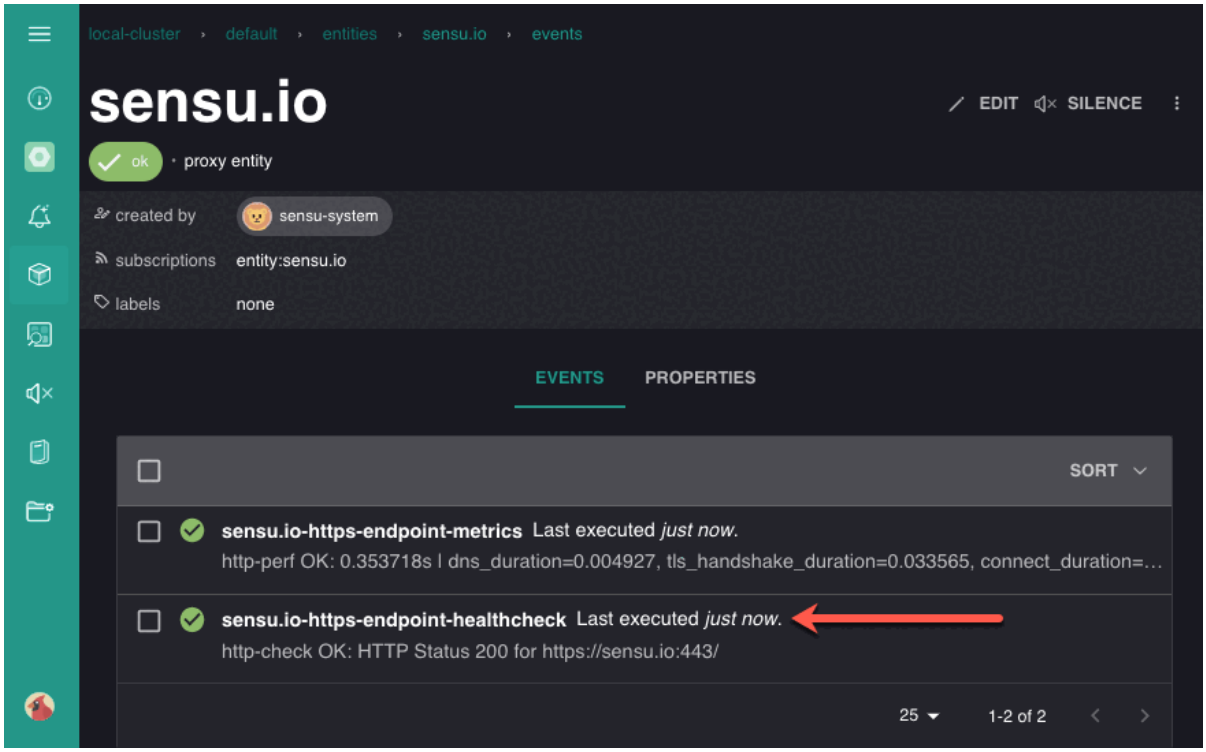
1. In the web UI, navigate to the Entities page.

2. Confirm that the Entities page lists a proxy entity named `sensu.io`.



3. Click the `sensu.io` entity to open the individual entity page.

4. Confirm that the individual entity page for `sensu.io` lists an event for the `sensu.io-https-endpoint-healthcheck` check.Click the event for details and history.



Use a proxy entity to monitor a website (command line configuration)

In this section, you'll use sensuctl to configure a check with a **proxy entity name** to monitor the status

of sensu.io so that Sensu creates an entity that represents the site and reports the status of the site under this entity.

Configure a Sensu entity

To run the check, you'll need a Sensu agent entity with the subscription `run_proxies`. Use sensuctl to add the `run_proxies` subscription to the entity the Sensu agent is observing.

> ***NOTE**: To find your entity name, run `sensuctl entity list`. The `ID` is the name of your entity.*

Before you run the following code, replace `<ENTITY_NAME>` with the name of the entity on your system.

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `run_proxies` and press enter.

Before you continue, confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register dynamic runtime asset

To power the check, you'll use the sensu/http-checks dynamic runtime asset. This community-tier asset includes the http status check command that your check will rely on.

Use `sensuctl asset add` to register the dynamic runtime asset:

```
sensuctl asset add sensu/http-checks:0.5.0 -r http-checks
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/http-checks:0.5.0
added asset: sensu/http-checks:0.5.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["http-checks"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `http-checks`.

You can also download the dynamic runtime asset definition from Bonsai and register the asset with `sensuctl create --file filename.yml` or `sensuctl create --file filename.json`.

Use sensuctl to confirm that the dynamic runtime asset is ready to use:

```
sensuctl asset list
```

The response should list the sensu/http-checks dynamic runtime asset (renamed to `http-checks`):

Name	URL	Hash
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_windows_amd64.tar.gz	52ae075
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_darwin_amd64.tar.gz	72d0f15
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_armv7.tar.gz	ef18587
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_arm64.tar.gz	3504ddf
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_386.tar.gz	60b8883
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_amd64.tar.gz	1db73a8

NOTE: *Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Create the check

Now that the dynamic runtime asset is registered, you can create a check named `check-sensu-site` to run the command `http-check --url https://sensu.io` with the sensu/http-checks dynamic runtime asset, at an interval of 15 seconds, for all agents subscribed to the `run_proxies` subscription, using the `sensu-site` proxy entity name.

The check includes the `round_robin` attribute set to `true` to distribute the check execution across all agents subscribed to the `run_proxies` subscription and avoid duplicate events.

To create the `check-sensu-site` check, run:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check-sensu-site
spec:
  command: http-check --url https://sensu.io
  interval: 15
  proxy_entity_name: sensu-site
  publish: true
  round_robin: true
  runtime_assets:
  - http-checks
  subscriptions:
  - run_proxies
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
```

```
      "name": "check-sensu-site"
    },
    "spec": {
      "command": "http-check --url https://sensu.io",
      "interval": 15,
      "proxy_entity_name": "sensu-site",
      "publish": true,
      "round_robin": true,
      "runtime_assets": [
        "http-checks"
      ],
      "subscriptions": [
        "run_proxies"
      ]
    }
  }
  EOF
```

Use sensuctl to confirm that Sensu added the check:

```
sensuctl check list
```

The response should list `check-sensu-site`:

```
       Name                  Command              Interval   Cron   Timeout   TTL   Subscriptions   Handlers     Assets      Hooks   Publish?   Stdin?   Metric Format   Metric Handlers
 ─────────────────── ───────────────────────────────── ────────── ──────── ───────── ────── ─────────────── ────────── ────────────── ─────── ────────── ──────── ─────────────── ─────────────────
  check-sensu-site   http-check --url https://sensu.io        15                0     0   proxy                       http-checks          true       false
```

Validate the check

Use sensuctl to confirm that Sensu created `sensu-site`. It might take a few moments for Sensu to execute the check and create the proxy entity.

```
sensuctl entity list
```

The response should list the `sensu-site` proxy entity:

```
     ID        Class    OS        Subscriptions               Last Seen
_____________ _______ _______ ____________________________ _____________________________________

 sensu-centos  agent   linux   proxy,entity:sensu-centos   2021-10-21 19:20:04 +0000 UTC
 sensu-site    proxy           entity:sensu-site           N/A
```

Then, use sensuctl to confirm that Sensu is monitoring `sensu-site` with the `check-sensu-site` check:

```
sensuctl event info sensu-site check-sensu-site
```

The response should list `check-sensu-site` status and history data for the `sensu-site` proxy entity:

```
=== sensu-site - check-sensu-site
Entity:    sensu-site
Check:     check-sensu-site
Output:    http-check OK: HTTP Status 200 for https://sensu.io
Status:    0
History:   0
Silenced:  false
Timestamp: 2021-10-21 19:20:06 +0000 UTC
UUID:      xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

You can also view the new proxy entity in your Sensu web UI.

Use proxy requests to monitor a group of websites

(command line configuration)

Suppose that instead of monitoring just sensu.io, you want to monitor multiple sites, like docs.sensu.io, packagecloud.io, and github.com.In this section, you'll use sensuctl to configure the `proxy_requests` check attribute, entity labels, and token substitution required to monitor three sites with the same check.

Before you start, register the http-checks dynamic runtime asset if you haven't already.

Create proxy entities

Instead of creating a proxy entity using the `proxy_entity_name` check attribute, use sensuctl to create proxy entities to represent the three sites you want to monitor.Your proxy entities need the `entity_class` attribute set to `proxy` to mark them as proxy entities as well as a few custom `labels` to identify them as a group and pass in individual URLs.

To add the proxy entity definitions, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Entity
api_version: core/v2
metadata:
  name: sensu-docs
  labels:
    proxy_type: website
    url: https://docs.sensu.io
spec:
  entity_class: proxy
---
type: Entity
api_version: core/v2
metadata:
  name: packagecloud-site
  labels:
    proxy_type: website
    url: https://packagecloud.io
spec:
  entity_class: proxy
```

```
---
type: Entity
api_version: core/v2
metadata:
  name: github-site
  labels:
    proxy_type: website
    url: https://github.com
spec:
  entity_class: proxy
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-docs",
    "labels": {
      "proxy_type": "website",
      "url": "https://docs.sensu.io"
    }
  },
  "spec": {
    "entity_class": "proxy"
  }
}
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "packagecloud-site",
    "labels": {
      "proxy_type": "website",
      "url": "https://packagecloud.io"
    }
  },
  "spec": {
    "entity_class": "proxy"
  }
```

```
}
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "github-site",
    "labels": {
      "proxy_type": "website",
      "url": "https://github.com"
    }
  },
  "spec": {
    "entity_class": "proxy"
  }
}
EOF
```

***PRO TIP**: When you create proxy entities, you can add any custom labels that make sense for your environment. For example, when monitoring a group of routers, you may want to add `ip_address` labels.*

Use sensuctl to confirm that the entities were added:

```
sensuctl entity list
```

The response should list the new `sensu-docs`, `packagecloud-site`, and `github-site` proxy entities:

```
      ID          Class    OS        Subscriptions               Last Seen
─────────────────── ──────── ──────── ──────────────────────────── ──────────────────────────────
 github-site        proxy                                N/A
 packagecloud-site  proxy                                N/A
 sensu-centos       agent    linux    proxy,entity:sensu-centos   2021-10-21 19:23:04 +0000 UTC
 sensu-docs         proxy                                N/A
 sensu-site         proxy    entity:sensu-site           N/A
```

Create a reusable HTTP check

Now that you have three proxy entities set up, each with a `proxy_type` and `url` label, you can use proxy requests and token substitution to create a single check that monitors all three sites.

The check includes the `round_robin` attribute set to `true` to distribute the check execution across all agents subscribed to the `run_proxies` subscription and avoid duplicate events.

To create the following check definition, run:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check-http
spec:
  command: 'http-check --url {{ .labels.url }}'
  interval: 15
  proxy_requests:
    entity_attributes:
      - entity.entity_class == 'proxy'
      - entity.labels.proxy_type == 'website'
  publish: true
  round_robin: true
  runtime_assets:
    - http-checks
  subscriptions:
    - run_proxies
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "check-http"
  },
  "spec": {
    "command": "http-check --url {{ .labels.url }}",
    "interval": 15,
    "proxy_requests": {
      "entity_attributes": [
        "entity.entity_class == 'proxy'",
        "entity.labels.proxy_type == 'website'"
      ]
    },
    "publish": true,
    "runtime_assets": [
      "http-checks"
    ],
    "round_robin": true,
    "subscriptions": [
      "run_proxies"
    ]
  }
}
EOF
```

Your `check-http` check uses the `proxy_requests` attribute to specify the applicable entities. In this case, you want to run the `check-http` check on all entities of entity class `proxy` and proxy type `website`. Because you're using this check to monitor multiple sites, the check command uses token substitution to apply the correct `url`.

Use sensuctl to confirm that Sensu created the check:

```
sensuctl check list
```

The response should include the `check-http` check:

```
     Name              Command                Interval   Cron   Timeout   TTL   Subscriptions   Handlers   Assets   Hooks
Publish?   Stdin?   Metric Format   Metric Handlers
_____________________ ______________________________________ __________ ______ _________ _____ _____________ __________ ________ ________
```

```
———— ———————— ———————— ———————— ———————— ———————— ————————
———————— ————————————
 check-http        http-check --url {{ .labels.url }}        15             0     0   proxy                 http-checks          true       false
 check-sensu-site   http-check --url https://sensu.io         15             0     0   proxy                 http-checks          true
false
```

Validate the check

Before you validate the check, make sure that you've registered the sensu/http-checks dynamic runtime asset and added the `run_proxies` subscription to a Sensu agent.

Use sensuctl to confirm that Sensu is monitoring docs.sensu.io, packagecloud.io, and github.com with the `check-http` check, returning a status of `0` (OK):

```
sensuctl event list
```

The response should list check status data for the `sensu-docs`, `packagecloud-site`, and `github-site` proxy entities:

```
     Entity            Check                          Output                          Status   Silenced          Timestamp
UUID
———————————————————— ————————————————————
————————————————————————————————————————————————————————————————————————————————— ————————
———————————— ———————————————————————————————— ————————————————————————————————————————————————
 github-site          check-http          http-check OK: HTTP Status 200 for https://github.com                    0    false      2021-10-
21 19:27:04 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

 packagecloud-site   check-http          http-check OK: HTTP Status 200 for https://packagecloud.io                0    false
2021-10-21 19:27:04 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

 sensu-centos         keepalive           Keepalive last sent from sensu-centos at 2021-10-21 19:27:44 +0000 UTC      0    false
2021-10-21 19:27:44 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 sensu-docs           check-http          http-check OK: HTTP Status 200 for https://docs.sensu.io                  0    false      2021-
10-21 19:27:03 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

 sensu-site           check-sensu-site   http-check OK: HTTP Status 200 for https://sensu.io                       0    false      2021-10-
21 19:27:05 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Next steps

The files you created with check and entity definitions can become part of your monitoring as code repository. Storing your Sensu configurations the same way you would store code means they are portable and repeatable. Monitoring as code makes it possible to move to a more robust deployment without losing what you've started here and reproduce one environment's configuration in another.

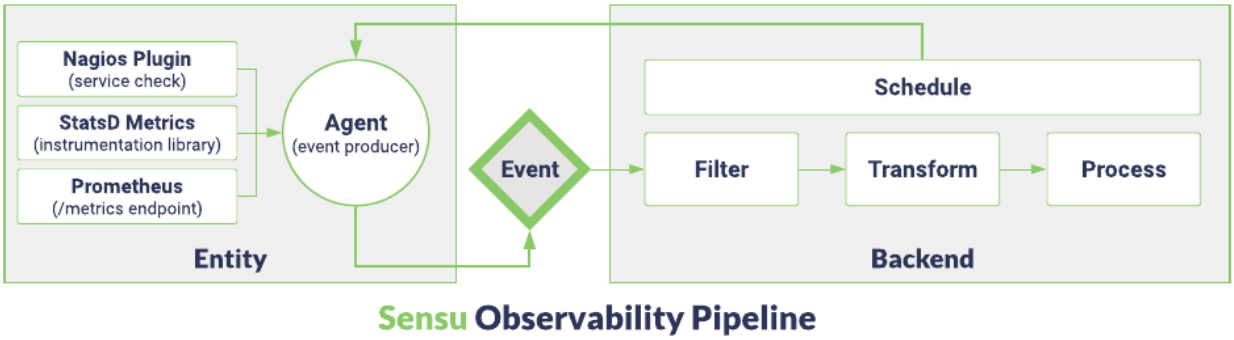
Now that you know how to run a proxy check to verify website status and use proxy requests to run a check on two different proxy entities based on label evaluation, you can receive alerts based on the events your checks create. Configure three more Sensu resources to start receiving alerts:

- Event filters, which the Sensu backend will apply to the observation data in events. Sensu then sends any events the filters do not remove for processing.
- Handlers, which process the events that filters do not remove.
- Pipelines, which are Sensu resources composed of observation event processing workflows made up of filters, mutators, and handlers. When you list a pipeline in a check definition, all the observability events that the check produces will be processed according to the pipeline's workflows.

Follow any of these guides to learn how to configure event filters, handlers, and pipelines and start sending alerts based on event data:

- Send email alerts with a pipeline
- Send PagerDuty alerts with Sensu
- Send Slack alerts with a pipeline

Events



Sensu Observability Pipeline

Next **or click any element in the pipeline to jump to it.**

Events are generic containers that Sensu uses to provide context to status and metrics check results. The context, called observation data, is information about the originating entity and the corresponding status or metric check result.

These generic containers allow Sensu to handle different types of events in the pipeline for comprehensive system and service monitoring and observability. Events can contain CPU, memory, and disk usage data; custom application metrics; log data you can send to an external database; and more.

Events require a timestamp, entity, and check. Each event must contain a check result, whether status or metrics. In certain cases, an event can contain both. Because events are polymorphic in nature, it is important to never assume their content (or lack of content).

Here's an example event that includes both status and metrics data, retrieved with sensuctl event info:

YML

```
---
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
```

```
    command: http-check --url http://localhost && http-perf --url http://localhost
      --warning 1s --critical 2s
    duration: 0.022274319
    env_vars: null
    executed: 1635959379
    handlers:
    - debug
    high_flap_threshold: 0
    history:
    - executed: 1635952820
      status: 0
    - executed: 1635952835
      status: 0
    - executed: 1635952850
      status: 0
    - executed: 1635952865
      status: 0
    - executed: 1635952880
      status: 0
    interval: 5
    is_silenced: false
    issued: 1635952880
    last_ok: 1635952880
    low_flap_threshold: 0
    metadata:
      name: collect-metrics
      namespace: default
    occurrences: 5
    occurrences_watermark: 5
    output: |
      http-check OK: HTTP Status 200 for http://localhost
      http-perf OK: 0.001150s | dns_duration=0.000257,
tls_handshake_duration=0.000000, connect_duration=0.000088,
first_byte_duration=0.001131, total_request_duration=0.001150
    output_metric_format: nagios_perfdata
    output_metric_handlers: null
    pipelines: []
    processed_by: sensu-centos
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
```

```
    - http-checks
    scheduler: memory
    secrets: null
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - webserver
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1635959379
    metadata:
      created_by: admin
      name: sensu-centos
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
    sensu_agent_version: 6.5.4
    subscriptions:
    - system
    - entity:sensu-centos
    - webserver
    system:
      arch: amd64
      cloud_provider: ""
      hostname: sensu-centos
      libc_type: glibc
      network:
```

```
        interfaces:
        - addresses:
          - 127.0.0.1/8
          - ::1/128
          name: lo
        - addresses:
          - 10.0.2.15/24
          - fe80::20b8:8cea:fa4:2e57/64
          mac: 08:00:27:8b:c9:3f
          name: eth0
        - addresses:
          - 192.168.200.95/24
          - fe80::a00:27ff:fe40:ab31/64
          mac: 08:00:27:40:ab:31
          name: eth1
      os: linux
      platform: centos
      platform_family: rhel
      platform_version: 7.9.2009
      processes: null
      vm_role: guest
      vm_system: vbox
    user: agent
  id: 12545deb-0e0f-480f-addf-34545d5a01c6
  pipelines: null
  sequence: 5
  timestamp: 1635952880
```

JSON

```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "http-check --url http://localhost && http-perf --url
http://localhost --warning 1s --critical 2s",
      "duration": 0.022274319,
```

```
      "env_vars": null,
      "executed": 1635959379,
      "handlers": [
        "debug"
      ],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1635952820,
          "status": 0
        },
        {
          "executed": 1635952835,
          "status": 0
        },
        {
          "executed": 1635952850,
          "status": 0
        },
        {
          "executed": 1635952865,
          "status": 0
        },
        {
          "executed": 1635952880,
          "status": 0
        }
      ],
      "interval": 5,
      "is_silenced": false,
      "issued": 1635952880,
      "last_ok": 1635952880,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "collect-metrics",
        "namespace": "default"
      },
      "occurrences": 5,
      "occurrences_watermark": 5,
      "output": "http-check OK: HTTP Status 200 for http://localhost\nhttp-perf OK:
0.001150s | dns_duration=0.000257, tls_handshake_duration=0.000000,
connect_duration=0.000088, first_byte_duration=0.001131,
```

```
total_request_duration=0.001150\n",
      "output_metric_format": "nagios_perfdata",
      "output_metric_handlers": null,
      "pipelines": [],
      "processed_by": "sensu-centos",
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "http-checks"
      ],
      "scheduler": "memory",
      "secrets": null,
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "webserver"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
    },
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1635959379,
      "metadata": {
        "created_by": "admin",
        "name": "sensu-centos",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
```

```
      "private_key",
      "secret"
    ],
    "sensu_agent_version": "6.5.4",
    "subscriptions": [
      "system",
      "entity:sensu-centos",
      "webserver"
    ],
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "sensu-centos",
      "libc_type": "glibc",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              ":1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::20b8:8cea:fa4:2e57/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          },
          {
            "addresses": [
              "192.168.200.95/24",
              "fe80::a00:27ff:fe40:ab31/64"
            ],
            "mac": "08:00:27:40:ab:31",
            "name": "eth1"
          }
        ]
      },
      "os": "linux",
```

```
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.9.2009",
        "processes": null,
        "vm_role": "guest",
        "vm_system": "vbox"
      },
      "user": "agent"
    },
    "id": "12545deb-0e0f-480f-addf-34545d5a01c6",
    "pipelines": null,
    "sequence": 5,
    "timestamp": 1635952880
  }
}
```


***NOTE**: Metrics data points are not included in events retrieved with sensuctl event info — these events include check output text rather than a set of metrics points. To view metrics points data as shown in the event below, add a debug handler that prints events to a JSON file.*


```
{
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.9.2009",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
```

```
          "name": "eth0",
          "mac": "08:00:27:8b:c9:3f",
          "addresses": [
            "10.0.2.15/24",
            "fe80::20b8:8cea:fa4:2e57/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:40:ab:31",
          "addresses": [
            "192.168.200.95/24",
            "fe80::a00:27ff:fe40:ab31/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "system",
    "entity:sensu-centos",
    "webserver"
  ],
  "last_seen": 1635952880,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
```

```
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin"
    },
    "sensu_agent_version": "6.5.4"
  },
  "check": {
    "command": "http-check --url http://localhost \\u0026\\u0026 http-perf --url
http://localhost --warning 1s --critical 2s",
    "handlers": [
      "debug"
    ],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "http-checks"
    ],
    "subscriptions": [
      "webserver"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 0.018747388,
    "executed": 1635952880,
    "history": [
      {
        "status": 0,
        "executed": 1635952820
      },
      {
        "status": 0,
        "executed": 1635952835
```

```
      },
      {
        "status": 0,
        "executed": 1635952850
      },
      {
        "status": 0,
        "executed": 1635952865
      },
      {
        "status": 0,
        "executed": 1635952880
      }
    ],
    "issued": 1635952880,
    "output": "http-check OK: HTTP Status 200 for http://localhost\nhttp-perf OK:
0.001059s | dns_duration=0.000235, tls_handshake_duration=0.000000,
connect_duration=0.000083, first_byte_duration=0.001040,
total_request_duration=0.001059\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1635952880,
    "occurrences": 5,
    "occurrences_watermark": 5,
    "output_metric_format": "nagios_perfdata",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "collect-metrics",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "scheduler": "memory",
    "processed_by": "sensu-centos",
    "pipelines": []
  },
  "metrics": {
    "handlers": null,
    "points": [
      {
```

```
        "name": "dns_duration",
        "value": 0.000235,
        "timestamp": 1635952880,
        "tags": null
      },
      {
        "name": "tls_handshake_duration",
        "value": 0,
        "timestamp": 1635952880,
        "tags": null
      },
      {
        "name": "connect_duration",
        "value": 0.000083,
        "timestamp": 1635952880,
        "tags": null
      },
      {
        "name": "first_byte_duration",
        "value": 0.00104,
        "timestamp": 1635952880,
        "tags": null
      },
      {
        "name": "total_request_duration",
        "value": 0.001059,
        "timestamp": 1635952880,
        "tags": null
      }
    ]
  },
  "metadata": {
    "namespace": "default"
  },
  "id": "7cde3e3f-beee-408f-b89a-1edccd0d3edb",
  "sequence": 5,
  "pipelines": null,
  "timestamp": 1635952880
}
```

Checks

Checks work with the Sensu agent to produce events automatically. You can use checks to monitor server resources, services, and application health as well as collect and analyze metrics.Checks define how Sensu will process events, as well as when and where events are generated via subscriptions and scheduling.

Read Monitor server resources to learn more about using checks to generate events.

Status-only events

A Sensu event is created every time a check result is processed by the Sensu server, regardless of the status the result indicates.The agent creates an event upon receipt of the check execution result and executes any configured hooks the check might have.From there, the status result is forwarded to the Sensu backend, where it is filtered, transformed, and processed.Potentially noteworthy events may be processed by one or more event handlers, for example to send an email or invoke an automated action.

Metrics-only events

Sensu events can be created when the agent receives metrics through the StatsD listener.The agent will translate the StatsD metrics to Sensu metric format and place them inside an event.Because these events do not contain checks, they bypass the store and are sent to the event pipeline and corresponding event handlers.

Status and metrics events

Events that contain *both* a check and metrics most likely originated from check output metric extraction. If a check is configured for metric extraction, the agent will parse the check output and transform it to Sensu metric format.Both the check results and resulting (extracted) metrics are stored inside the event Event handlers from `event.Check.Handlers` and `event.Metrics.Handlers` will be invoked.

Proxy entities and events

You can create events with proxy entities, which are dynamically created entities that Sensu adds to

the entity store if an entity does not already exist for a check result. Proxy entities allow Sensu to monitor external resources on systems where you cannot install a Sensu agent, like a network switch or website. Read Monitor external resources to learn how to use a proxy entity to monitor a website.

core/v2/events API endpoints

Sensu's core/v2/events API endpoints provide HTTP access to create, retrieve, update, and delete events. If you create a new event that references an entity that does not already exist, the Sensu backend will automatically create a proxy entity when the event is published.

Events reference

An event is a generic container used by Sensu to provide context to checks and metrics. The context, called observation data or event data, contains information about the originating entity and the corresponding check or metric result. An event must contain a status or metrics. In certain cases, an event can contain both a status and metrics. These generic containers allow Sensu to handle different types of events in the observability pipeline. Because events are polymorphic in nature, it is important to never assume their contents (or lack of content).

Event format

Sensu events contain:

- `entity` scope (required)
  - Information about the source of the event, including any attributes defined in the entity specification
- `check` scope (optional if the `metrics` scope is present)
  - Information about how the event was created, including any attributes defined in the check specification
  - Information about the event and its history, including any check attributes defined in the event specification on this page
- `metrics` scope (optional if the `check` scope is present)
  - Metric points in Sensu metric format
- `timestamp`
  - Time that the event occurred in seconds since the Unix epoch
- `id`
  - Universally unique identifier (UUID) for the event (logged as `event_id`)

Example status-only event

The following example shows the complete resource definition for a status-only event:

YML

```
---
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
    command: check-cpu-usage -w 75 -c 90
    duration: 5.058211427
    env_vars: null
    executed: 1617050501
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1617050261
      status: 0
    - executed: 1617050321
      status: 0
    - executed: 1617050381
      status: 0
    - executed: 1617050441
      status: 0
    - executed: 1617050501
      status: 0
    interval: 60
    is_silenced: false
    processed_by: sensu-centos
    issued: 1617050501
    last_ok: 1617050501
    low_flap_threshold: 0
    metadata:
      name: check_cpu
      namespace: default
    occurrences: 5
    occurrences_watermark: 5
    output: |
            CheckCPU TOTAL OK: total=0.41 user=0.2 nice=0.0 system=0.2 idle=99.59
```

```
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0
    output_metric_format: ""
    output_metric_handlers: null
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
    - check-cpu-usage
    scheduler: memory
    secrets: null
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1617050501
    metadata:
      name: sensu-centos
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
    sensu_agent_version: 6.2.6
    subscriptions:
    - linux
    - entity:sensu-centos
    system:
```

```
      arch: amd64
      cloud_provider: ""
      hostname: sensu-centos
      libc_type: glibc
      network:
        interfaces:
        - addresses:
          - 127.0.0.1/8
          - ::1/128
          name: lo
        - addresses:
          - 10.0.2.15/24
          - fe80::a268:dcce:3be:1c73/64
          mac: 08:00:27:8b:c9:3f
          name: eth0
        - addresses:
          - 172.28.128.45/24
          - fe80::a00:27ff:feb2:dc46/64
          mac: 08:00:27:b2:dc:46
          name: eth1
      os: linux
      platform: centos
      platform_family: rhel
      platform_version: 7.5.1804
      processes: null
      vm_role: guest
      vm_system: vbox
    user: agent
  pipelines:
    - api_version: core/v2
      type: Pipeline
      name: incident_alerts
  id: 3c3e68f6-6db7-40d3-9b84-4d61817ae559
  sequence: 5
  timestamp: 1617050507
```

JSON

```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
```

```
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "check-cpu-usage -w 75 -c 90",
      "duration": 5.058211427,
      "env_vars": null,
      "executed": 1617050501,
      "handlers": [],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1617050261,
          "status": 0
        },
        {
          "executed": 1617050321,
          "status": 0
        },
        {
          "executed": 1617050381,
          "status": 0
        },
        {
          "executed": 1617050441,
          "status": 0
        },
        {
          "executed": 1617050501,
          "status": 0
        }
      ],
      "interval": 60,
      "is_silenced": false,
      "processed_by": "sensu-centos",
      "issued": 1617050501,
      "last_ok": 1617050501,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
```

```
      },
      "occurrences": 5,
      "occurrences_watermark": 5,
      "output": "CheckCPU TOTAL OK: total=0.41 user=0.2 nice=0.0 system=0.2
idle=99.59 iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "output_metric_format": "",
      "output_metric_handlers": null,
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "scheduler": "memory",
      "secrets": null,
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
    },
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1617050501,
      "metadata": {
        "name": "sensu-centos",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
```

```
      "secret_key",
      "private_key",
      "secret"
    ],
    "sensu_agent_version": "6.2.6",
    "subscriptions": [
      "linux",
      "entity:sensu-centos"
    ],
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "sensu-centos",
      "libc_type": "glibc",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              ":1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::a268:dcce:3be:1c73/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          },
          {
            "addresses": [
              "172.28.128.45/24",
              "fe80::a00:27ff:feb2:dc46/64"
            ],
            "mac": "08:00:27:b2:dc:46",
            "name": "eth1"
          }
        ]
      },
      "os": "linux",
```

```
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "processes": null,
        "vm_role": "guest",
        "vm_system": "vbox"
      },
      "user": "agent"
    },
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ],
    "id": "3c3e68f6-6db7-40d3-9b84-4d61817ae559",
    "sequence": 5,
    "timestamp": 1617050507
  }
}
```


Example status-only event from the Sensu API

Sensu sends events to the backend in `json` format, without the outer-level `spec` wrapper or `type` and `api_version` attributes that are included in the `wrapped-json` format. This is the format that events are in when Sensu sends them to the observability pipeline for processing:


```
{
  "check": {
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
```

```
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 5.058211427,
    "executed": 1617050501,
    "history": [
      {
        "status": 0,
        "executed": 1617050261
      },
      {
        "status": 0,
        "executed": 1617050321
      },
      {
        "status": 0,
        "executed": 1617050381
      },
      {
        "status": 0,
        "executed": 1617050441
      },
      {
        "status": 0,
        "executed": 1617050501
      }
    ],
    "issued": 1617050501,
    "output": "CheckCPU TOTAL OK: total=0.4 user=0.2 nice=0.0 system=0.2 idle=99.6
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1617050501,
    "occurrences": 5,
```

```
    "occurrences_watermark": 5,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "processed_by": "sensu-centos",
    "scheduler": "memory"
  },
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::a268:dcce:3be:1c73/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:b2:dc:46",
            "addresses": [
```

```
            "172.28.128.45/24",
            "fe80::a00:27ff:feb2:dc46/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "linux",
    "entity:sensu-centos"
  ],
  "last_seen": 1617049781,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default"
  },
  "sensu_agent_version": "6.2.6"
},
"pipelines": [
  {
    "api_version": "core/v2",
    "type": "Pipeline",
```

```
      "name": "incident_alerts"
    }
  ],
  "id": "3c3e68f6-6db7-40d3-9b84-4d61817ae559",
  "metadata": {
    "namespace": "default"
  },
  "sequence": 5,
  "timestamp": 1617050507
}
```


Example metrics-only event

This example shows a complete metrics-only event, retrieved with sensuctl event info:

YML


```
---
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
    command: system-check
    duration: 3.012411959
    env_vars: null
    executed: 1635959903
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1635959873
      status: 0
    - executed: 1635959883
      status: 0
    - executed: 1635959893
      status: 0
    - executed: 1635959903
      status: 0
```

```
    interval: 10
    is_silenced: false
    issued: 1635959903
    last_ok: 1635959903
    low_flap_threshold: 0
    metadata:
      labels:
        sensu.io/managed_by: sensuctl
      name: system-check
      namespace: default
    occurrences: 4
    occurrences_watermark: 4
    output: |+
      # HELP system_cpu_cores [GAUGE] Number of cpu cores on the system
      # TYPE system_cpu_cores GAUGE
      system_cpu_cores{} 1 1635959903645
      # HELP system_cpu_idle [GAUGE] Percent of time all cpus were idle
      # TYPE system_cpu_idle GAUGE
      system_cpu_idle{cpu="cpu0"} 98.94366197187135 1635959903645
      system_cpu_idle{cpu="cpu-total"} 98.94366197187135 1635959903645
      # HELP system_cpu_used [GAUGE] Percent of time all cpus were used
      # TYPE system_cpu_used GAUGE
      system_cpu_used{cpu="cpu0"} 1.0563380281286499 1635959903645
      system_cpu_used{cpu="cpu-total"} 1.0563380281286499 1635959903645
      # HELP system_cpu_user [GAUGE] Percent of time total cpu was used by normal
processes in user mode
      # TYPE system_cpu_user GAUGE
      system_cpu_user{cpu="cpu0"} 0.7042253521124505 1635959903645
      system_cpu_user{cpu="cpu-total"} 0.7042253521124505 1635959903645
      # HELP system_cpu_system [GAUGE] Percent of time all cpus used by processes
executed in kernel mode
      # TYPE system_cpu_system GAUGE
      system_cpu_system{cpu="cpu0"} 0.35211267605672564 1635959903645
      system_cpu_system{cpu="cpu-total"} 0.35211267605672564 1635959903645
      # HELP system_cpu_nice [GAUGE] Percent of time all cpus used by niced
processes in user mode
      # TYPE system_cpu_nice GAUGE
      system_cpu_nice{cpu="cpu0"} 0 1635959903645
      system_cpu_nice{cpu="cpu-total"} 0 1635959903645
      # HELP system_cpu_iowait [GAUGE] Percent of time all cpus waiting for I/O to
complete
      # TYPE system_cpu_iowait GAUGE
```

```
      system_cpu_iowait{cpu="cpu0"} 0 1635959903645
      system_cpu_iowait{cpu="cpu-total"} 0 1635959903645
      # HELP system_cpu_irq [GAUGE] Percent of time all cpus servicing interrupts
      # TYPE system_cpu_irq GAUGE
      system_cpu_irq{cpu="cpu0"} 0 1635959903645
      system_cpu_irq{cpu="cpu-total"} 0 1635959903645
      # HELP system_cpu_sortirq [GAUGE] Percent of time all cpus servicing software
interrupts
      # TYPE system_cpu_sortirq GAUGE
      system_cpu_sortirq{cpu="cpu0"} 0 1635959903645
      system_cpu_sortirq{cpu="cpu-total"} 0 1635959903645
      # HELP system_cpu_stolen [GAUGE] Percent of time all cpus serviced virtual
hosts operating systems
      # TYPE system_cpu_stolen GAUGE
      system_cpu_stolen{cpu="cpu0"} 0 1635959903645
      system_cpu_stolen{cpu="cpu-total"} 0 1635959903645
      # HELP system_cpu_guest [GAUGE] Percent of time all cpus serviced guest
operating system
      # TYPE system_cpu_guest GAUGE
      system_cpu_guest{cpu="cpu0"} 0 1635959903645
      system_cpu_guest{cpu="cpu-total"} 0 1635959903645
      # HELP system_cpu_guest_nice [GAUGE] Percent of time all cpus serviced niced
guest operating system
      # TYPE system_cpu_guest_nice GAUGE
      system_cpu_guest_nice{cpu="cpu0"} 0 1635959903645
      system_cpu_guest_nice{cpu="cpu-total"} 0 1635959903645
      # HELP system_mem_used [GAUGE] Percent of memory used
      # TYPE system_mem_used GAUGE
      system_mem_used{} 24.63435866529588 1635959903645
      # HELP system_mem_used_bytes [GAUGE] Used memory in bytes
      # TYPE system_mem_used_bytes GAUGE
      system_mem_used_bytes{} 2.56159744e+08 1635959903645
      # HELP system_mem_total_bytes [GAUGE] Total memory in bytes
      # TYPE system_mem_total_bytes GAUGE
      system_mem_total_bytes{} 1.039847424e+09 1635959903645
      # HELP system_swap_used [GAUGE] Percent of swap used
      # TYPE system_swap_used GAUGE
      system_swap_used{} 0.0976564362648702 1635959903645
      # HELP system_swap_used_bytes [GAUGE] Used swap in bytes
      # TYPE system_swap_used_bytes GAUGE
      system_swap_used_bytes{} 2.56159744e+08 1635959903645
      # HELP system_swap_total_bytes [GAUGE] Total swap in bytes
```

```
      # TYPE system_swap_total_bytes GAUGE
      system_swap_total_bytes{} 2.147479552e+09 1635959903645
      # HELP system_load_load1 [GAUGE] System load averaged over 1 minute, high load
value dependant on number of cpus in system
      # TYPE system_load_load1 GAUGE
      system_load_load1{} 0.09 1635959903645
      # HELP system_load_load5 [GAUGE] System load averaged over 5 minute, high load
value dependent on number of cpus in system
      # TYPE system_load_load5 GAUGE
      system_load_load5{} 0.04 1635959903645
      # HELP system_load_load15 [GAUGE] System load averaged over 15 minute, high
load value dependent on number of cpus in system
      # TYPE system_load_load15 GAUGE
      system_load_load15{} 0.05 1635959903645
      # HELP system_load_load1_per_cpu [GAUGE] System load averaged over 1 minute
normalized by cpu count, values > 1 means system may be overloaded
      # TYPE system_load_load1_per_cpu GAUGE
      system_load_load1_per_cpu{} 0.09 1635959903645
      # HELP system_load_load5_per_cpu [GAUGE] System load averaged over 5 minute
normalized by cpu count, values > 1 means system may be overloaded
      # TYPE system_load_load5_per_cpu GAUGE
      system_load_load5_per_cpu{} 0.04 1635959903645
      # HELP system_load_load15_per_cpu [GAUGE] System load averaged over 15 minute
normalized by cpu count, values > 1 means system may be overloaded
      # TYPE system_load_load15_per_cpu GAUGE
      system_load_load15_per_cpu{} 0.05 1635959903645
      # HELP system_host_uptime [COUNTER] Host uptime in seconds
      # TYPE system_host_uptime COUNTER
      system_host_uptime{} 15488 1635959903645
      # HELP system_host_processes [GAUGE] Number of host processes
      # TYPE system_host_processes GAUGE
      system_host_processes{} 112 1635959903645
    output_metric_format: prometheus_text
    output_metric_handlers: null
    pipelines: []
    processed_by: sensu-centos
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
    - system-check
    scheduler: memory
```

```
    secrets: null
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1635959903
    metadata:
      created_by: admin
      name: sensu-centos
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
    sensu_agent_version: 6.5.4
    subscriptions:
    - system
    - entity:sensu-centos
    - webserver
    system:
      arch: amd64
      cloud_provider: ""
      hostname: sensu-centos
      libc_type: glibc
      network:
        interfaces:
        - addresses:
```

```
          - 127.0.0.1/8
          - ::1/128
          name: lo
        - addresses:
          - 10.0.2.15/24
          - fe80::20b8:8cea:fa4:2e57/64
          mac: 08:00:27:8b:c9:3f
          name: eth0
        - addresses:
          - 192.168.200.95/24
          - fe80::a00:27ff:fe40:ab31/64
          mac: 08:00:27:40:ab:31
          name: eth1
      os: linux
      platform: centos
      platform_family: rhel
      platform_version: 7.9.2009
      processes: null
      vm_role: guest
      vm_system: vbox
    user: agent
  id: 07425e48-e163-47d3-8bc8-17fbaa27e469
  pipelines: null
  sequence: 122
  timestamp: 1635959906
```

JSON

```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "system-check",
      "duration": 3.012411959,
      "env_vars": null,
      "executed": 1635959903,
      "handlers": [],
```

```
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1635959873,
          "status": 0
        },
        {
          "executed": 1635959883,
          "status": 0
        },
        {
          "executed": 1635959893,
          "status": 0
        },
        {
          "executed": 1635959903,
          "status": 0
        }
      ],
      "interval": 10,
      "is_silenced": false,
      "issued": 1635959903,
      "last_ok": 1635959903,
      "low_flap_threshold": 0,
      "metadata": {
        "labels": {
          "sensu.io/managed_by": "sensuctl"
        },
        "name": "system-check",
        "namespace": "default"
      },
      "occurrences": 4,
      "occurrences_watermark": 4,
      "output": "# HELP system_cpu_cores [GAUGE] Number of cpu cores on the
system\n# TYPE system_cpu_cores GAUGE\nsystem_cpu_cores{} 1 1635959903645\n# HELP
system_cpu_idle [GAUGE] Percent of time all cpus were idle\n# TYPE system_cpu_idle
GAUGE\nsystem_cpu_idle{cpu=\"cpu0\"} 98.94366197187135
1635959903645\nsystem_cpu_idle{cpu=\"cpu-total\"} 98.94366197187135 1635959903645\n#
HELP system_cpu_used [GAUGE] Percent of time all cpus were used\n# TYPE
system_cpu_used GAUGE\nsystem_cpu_used{cpu=\"cpu0\"} 1.0563380281286499
1635959903645\nsystem_cpu_used{cpu=\"cpu-total\"} 1.0563380281286499
1635959903645\n# HELP system_cpu_user [GAUGE] Percent of time total cpu was used by
```

normal processes in user mode\n# TYPE system_cpu_user
GAUGE\nsystem_cpu_user{cpu=\"cpu0\"} 0.7042253521124505
1635959903645\nsystem_cpu_user{cpu=\"cpu-total\"} 0.7042253521124505
1635959903645\n# HELP system_cpu_system [GAUGE] Percent of time all cpus used by
processes executed in kernel mode\n# TYPE system_cpu_system
GAUGE\nsystem_cpu_system{cpu=\"cpu0\"} 0.35211267605672564
1635959903645\nsystem_cpu_system{cpu=\"cpu-total\"} 0.35211267605672564
1635959903645\n# HELP system_cpu_nice [GAUGE] Percent of time all cpus used by niced
processes in user mode\n# TYPE system_cpu_nice GAUGE\nsystem_cpu_nice{cpu=\"cpu0\"}
0 1635959903645\nsystem_cpu_nice{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_cpu_iowait [GAUGE] Percent of time all cpus waiting for I/O to complete\n#
TYPE system_cpu_iowait GAUGE\nsystem_cpu_iowait{cpu=\"cpu0\"} 0
1635959903645\nsystem_cpu_iowait{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_cpu_irq [GAUGE] Percent of time all cpus servicing interrupts\n# TYPE
system_cpu_irq GAUGE\nsystem_cpu_irq{cpu=\"cpu0\"} 0
1635959903645\nsystem_cpu_irq{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_cpu_sortirq [GAUGE] Percent of time all cpus servicing software interrupts\n#
TYPE system_cpu_sortirq GAUGE\nsystem_cpu_sortirq{cpu=\"cpu0\"} 0
1635959903645\nsystem_cpu_sortirq{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_cpu_stolen [GAUGE] Percent of time all cpus serviced virtual hosts operating
systems\n# TYPE system_cpu_stolen GAUGE\nsystem_cpu_stolen{cpu=\"cpu0\"} 0
1635959903645\nsystem_cpu_stolen{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_cpu_guest [GAUGE] Percent of time all cpus serviced guest operating system\n#
TYPE system_cpu_guest GAUGE\nsystem_cpu_guest{cpu=\"cpu0\"} 0
1635959903645\nsystem_cpu_guest{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_cpu_guest_nice [GAUGE] Percent of time all cpus serviced niced guest
operating system\n# TYPE system_cpu_guest_nice
GAUGE\nsystem_cpu_guest_nice{cpu=\"cpu0\"} 0
1635959903645\nsystem_cpu_guest_nice{cpu=\"cpu-total\"} 0 1635959903645\n# HELP
system_mem_used [GAUGE] Percent of memory used\n# TYPE system_mem_used
GAUGE\nsystem_mem_used{} 24.63435866529588 1635959903645\n# HELP
system_mem_used_bytes [GAUGE] Used memory in bytes\n# TYPE system_mem_used_bytes
GAUGE\nsystem_mem_used_bytes{} 2.56159744e+08 1635959903645\n# HELP
system_mem_total_bytes [GAUGE] Total memory in bytes\n# TYPE system_mem_total_bytes
GAUGE\nsystem_mem_total_bytes{} 1.039847424e+09 1635959903645\n# HELP
system_swap_used [GAUGE] Percent of swap used\n# TYPE system_swap_used
GAUGE\nsystem_swap_used{} 0.0976564362648702 1635959903645\n# HELP
system_swap_used_bytes [GAUGE] Used swap in bytes\n# TYPE system_swap_used_bytes
GAUGE\nsystem_swap_used_bytes{} 2.56159744e+08 1635959903645\n# HELP
system_swap_total_bytes [GAUGE] Total swap in bytes\n# TYPE system_swap_total_bytes
GAUGE\nsystem_swap_total_bytes{} 2.147479552e+09 1635959903645\n# HELP
system_load_load1 [GAUGE] System load averaged over 1 minute, high load value

```
dependant on number of cpus in system\n# TYPE system_load_load1
GAUGE\nsystem_load_load1{} 0.09 1635959903645\n# HELP system_load_load5 [GAUGE]
System load averaged over 5 minute, high load value dependent on number of cpus in
system\n# TYPE system_load_load5 GAUGE\nsystem_load_load5{} 0.04 1635959903645\n#
HELP system_load_load15 [GAUGE] System load averaged over 15 minute, high load value
dependent on number of cpus in system\n# TYPE system_load_load15
GAUGE\nsystem_load_load15{} 0.05 1635959903645\n# HELP system_load_load1_per_cpu
[GAUGE] System load averaged over 1 minute normalized by cpu count, values > 1 means
system may be overloaded\n# TYPE system_load_load1_per_cpu
GAUGE\nsystem_load_load1_per_cpu{} 0.09 1635959903645\n# HELP
system_load_load5_per_cpu [GAUGE] System load averaged over 5 minute normalized by
cpu count, values > 1 means system may be overloaded\n# TYPE
system_load_load5_per_cpu GAUGE\nsystem_load_load5_per_cpu{} 0.04 1635959903645\n#
HELP system_load_load15_per_cpu [GAUGE] System load averaged over 15 minute
normalized by cpu count, values > 1 means system may be overloaded\n# TYPE
system_load_load15_per_cpu GAUGE\nsystem_load_load15_per_cpu{} 0.05 1635959903645\n#
HELP system_host_uptime [COUNTER] Host uptime in seconds\n# TYPE system_host_uptime
COUNTER\nsystem_host_uptime{} 15488 1635959903645\n# HELP system_host_processes
[GAUGE] Number of host processes\n# TYPE system_host_processes
GAUGE\nsystem_host_processes{} 112 1635959903645\n\n",
      "output_metric_format": "prometheus_text",
      "output_metric_handlers": null,
      "pipelines": [],
      "processed_by": "sensu-centos",
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "system-check"
      ],
      "scheduler": "memory",
      "secrets": null,
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
```

```
  },
  "entity": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "agent",
    "last_seen": 1635959903,
    "metadata": {
      "created_by": "admin",
      "name": "sensu-centos",
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "sensu_agent_version": "6.5.4",
    "subscriptions": [
      "system",
      "entity:sensu-centos",
      "webserver"
    ],
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "sensu-centos",
      "libc_type": "glibc",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              ":1/128"
            ],
            "name": "lo"
          },
```

```
            {
              "addresses": [
                "10.0.2.15/24",
                "fe80::20b8:8cea:fa4:2e57/64"
              ],
              "mac": "08:00:27:8b:c9:3f",
              "name": "eth0"
            },
            {
              "addresses": [
                "192.168.200.95/24",
                "fe80::a00:27ff:fe40:ab31/64"
              ],
              "mac": "08:00:27:40:ab:31",
              "name": "eth1"
            }
          ]
        },
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.9.2009",
        "processes": null,
        "vm_role": "guest",
        "vm_system": "vbox"
      },
      "user": "agent"
    },
    "id": "07425e48-e163-47d3-8bc8-17fbaa27e469",
    "pipelines": null,
    "sequence": 122,
    "timestamp": 1635959906
  }
}
```


Metrics data points are not included in events retrieved with sensuctl event info — those events include check output text rather than a set of metrics points.To view metrics points data as shown in the following example, create a pipeline workflow that includes a debug handler that prints events to a JSON file:

```
{
  "metrics": {
    "points": [
      {
        "name": "system_cpu_sortirq",
        "value": 0,
        "timestamp": 1635952533,
        "tags": [
          {
            "name": "cpu",
            "value": "cpu0"
          },
          {
            "name": "prom_type",
            "value": "gauge"
          }
        ]
      },
      {
        "name": "system_cpu_sortirq",
        "value": 0,
        "timestamp": 1635952533,
        "tags": [
          {
            "name": "cpu",
            "value": "cpu-total"
          },
          {
            "name": "prom_type",
            "value": "gauge"
          }
        ]
      },
      {
        "name": "system_cpu_guest",
        "value": 0,
        "timestamp": 1635952533,
        "tags": [
          {
            "name": "cpu",
            "value": "cpu0"
          },
```

```
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_guest",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu-total"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_mem_used_bytes",
      "value": 260579328,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_mem_total_bytes",
      "value": 1039847424,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
```

```
    },
    {
      "name": "system_swap_used",
      "value": 0.0736237976528123,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_used",
      "value": 0.6756756756291793,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_used",
      "value": 0.6756756756291793,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu-total"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
```

```
    {
      "name": "system_cpu_nice",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_nice",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu-total"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_irq",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
          "value": "gauge"
```

```
      }
    ]
  },
  {
    "name": "system_cpu_irq",
    "value": 0,
    "timestamp": 1635952533,
    "tags": [
      {
        "name": "cpu",
        "value": "cpu-total"
      },
      {
        "name": "prom_type",
        "value": "gauge"
      }
    ]
  },
  {
    "name": "system_load_load1",
    "value": 0.01,
    "timestamp": 1635952533,
    "tags": [
      {
        "name": "prom_type",
        "value": "gauge"
      }
    ]
  },
  {
    "name": "system_host_uptime",
    "value": 10642,
    "timestamp": 1635952533,
    "tags": [
      {
        "name": "prom_type",
        "value": "counter"
      }
    ]
  },
  {
    "name": "system_host_processes",
```

```
      "value": 116,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_load_load5_per_cpu",
      "value": 0.02,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_cores",
      "value": 1,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_swap_used_bytes",
      "value": 260579328,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
```

```
    {
      "name": "system_load_load5",
      "value": 0.02,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_mem_used",
      "value": 25.059381019344624,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_swap_total_bytes",
      "value": 2147479552,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_load_load1_per_cpu",
      "value": 0.01,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        }
```

```
    ]
  },
  {
    "name": "system_load_load15_per_cpu",
    "value": 0.05,
    "timestamp": 1635952533,
    "tags": [
      {
        "name": "prom_type",
        "value": "gauge"
      }
    ]
  },
  {
    "name": "system_cpu_idle",
    "value": 99.32432432437082,
    "timestamp": 1635952533,
    "tags": [
      {
        "name": "prom_type",
        "value": "gauge"
      },
      {
        "name": "cpu",
        "value": "cpu0"
      }
    ]
  },
  {
    "name": "system_cpu_idle",
    "value": 99.32432432437082,
    "timestamp": 1635952533,
    "tags": [
      {
        "name": "cpu",
        "value": "cpu-total"
      },
      {
        "name": "prom_type",
        "value": "gauge"
      }
    ]
```

```
    },
    {
      "name": "system_cpu_user",
      "value": 0.3378378378376302,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_user",
      "value": 0.3378378378376302,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu-total"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_iowait",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
```

```
            "value": "gauge"
          }
        ]
      },
      {
        "name": "system_cpu_iowait",
        "value": 0,
        "timestamp": 1635952533,
        "tags": [
          {
            "name": "cpu",
            "value": "cpu-total"
          },
          {
            "name": "prom_type",
            "value": "gauge"
          }
        ]
      },
      {
        "name": "system_load_load15",
        "value": 0.05,
        "timestamp": 1635952533,
        "tags": [
          {
            "name": "prom_type",
            "value": "gauge"
          }
        ]
      },
      {
        "name": "system_cpu_system",
        "value": 0.3378378378376302,
        "timestamp": 1635952533,
        "tags": [
          {
            "name": "prom_type",
            "value": "gauge"
          },
          {
            "name": "cpu",
            "value": "cpu0"
```

```
        }
      ]
    },
    {
      "name": "system_cpu_system",
      "value": 0.3378378378376302,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu-total"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_stolen",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_stolen",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "prom_type",
          "value": "gauge"
        },
```

```
        {
          "name": "cpu",
          "value": "cpu-total"
        }
      ]
    },
    {
      "name": "system_cpu_guest_nice",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu0"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "system_cpu_guest_nice",
      "value": 0,
      "timestamp": 1635952533,
      "tags": [
        {
          "name": "cpu",
          "value": "cpu-total"
        },
        {
          "name": "prom_type",
          "value": "gauge"
        }
      ]
    }
  ]
},
"metadata": {
  "namespace": "default"
},
"id": "afdeb823-74c2-4921-891a-465a2095cb5a",
```

```
  "sequence": 6,
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "debug_pipeline"
    }
  ],
  "timestamp": 1635952536,
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.9.2009",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::20b8:8cea:fa4:2e57/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:40:ab:31",
            "addresses": [
              "192.168.200.95/24",
              "fe80::a00:27ff:fe40:ab31/64"
            ]
          }
```

```
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:sensu-centos"
    ],
    "last_seen": 1635952533,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default"
    },
    "sensu_agent_version": "6.5.4"
  },
  "check": {
    "command": "system-check",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
```

```
      "system-check"
    ],
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 3.01062768,
    "executed": 1635952533,
    "history": [
      {
        "status": 0,
        "executed": 1635952283
      },
      {
        "status": 0,
        "executed": 1635952293
      },
      {
        "status": 0,
        "executed": 1635952303
      },
      {
        "status": 0,
        "executed": 1635952313
      },
      {
        "status": 0,
        "executed": 1635952421
      },
      {
        "status": 0,
        "executed": 1635952533
      }
    ],
    "issued": 1635952533,
    "output": "# HELP system_cpu_cores [GAUGE] Number of cpu cores on the system\n#
```

TYPE system_cpu_cores GAUGE\nsystem_cpu_cores{} 1 1635952533657\n# HELP system_cpu_idle [GAUGE] Percent of time all cpus were idle\n# TYPE system_cpu_idle GAUGE\nsystem_cpu_idle{cpu=\"cpu0\"} 99.32432432437082 1635952533657\nsystem_cpu_idle{cpu=\"cpu-total\"} 99.32432432437082 1635952533657\n# HELP system_cpu_used [GAUGE] Percent of time all cpus were used\n# TYPE system_cpu_used GAUGE\nsystem_cpu_used{cpu=\"cpu0\"} 0.6756756756291793 1635952533657\nsystem_cpu_used{cpu=\"cpu-total\"} 0.6756756756291793 1635952533657\n# HELP system_cpu_user [GAUGE] Percent of time total cpu was used by normal processes in user mode\n# TYPE system_cpu_user GAUGE\nsystem_cpu_user{cpu=\"cpu0\"} 0.3378378378376302 1635952533657\nsystem_cpu_user{cpu=\"cpu-total\"} 0.3378378378376302 1635952533657\n# HELP system_cpu_system [GAUGE] Percent of time all cpus used by processes executed in kernel mode\n# TYPE system_cpu_system GAUGE\nsystem_cpu_system{cpu=\"cpu0\"} 0.3378378378376302 1635952533657\nsystem_cpu_system{cpu=\"cpu-total\"} 0.3378378378376302 1635952533657\n# HELP system_cpu_nice [GAUGE] Percent of time all cpus used by niced processes in user mode\n# TYPE system_cpu_nice GAUGE\nsystem_cpu_nice{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_nice{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_cpu_iowait [GAUGE] Percent of time all cpus waiting for I/O to complete\n# TYPE system_cpu_iowait GAUGE\nsystem_cpu_iowait{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_iowait{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_cpu_irq [GAUGE] Percent of time all cpus servicing interrupts\n# TYPE system_cpu_irq GAUGE\nsystem_cpu_irq{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_irq{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_cpu_sortirq [GAUGE] Percent of time all cpus servicing software interrupts\n# TYPE system_cpu_sortirq GAUGE\nsystem_cpu_sortirq{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_sortirq{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_cpu_stolen [GAUGE] Percent of time all cpus serviced virtual hosts operating systems\n# TYPE system_cpu_stolen GAUGE\nsystem_cpu_stolen{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_stolen{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_cpu_guest [GAUGE] Percent of time all cpus serviced guest operating system\n# TYPE system_cpu_guest GAUGE\nsystem_cpu_guest{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_guest{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_cpu_guest_nice [GAUGE] Percent of time all cpus serviced niced guest operating system\n# TYPE system_cpu_guest_nice GAUGE\nsystem_cpu_guest_nice{cpu=\"cpu0\"} 0 1635952533657\nsystem_cpu_guest_nice{cpu=\"cpu-total\"} 0 1635952533657\n# HELP system_mem_used [GAUGE] Percent of memory used\n# TYPE system_mem_used GAUGE\nsystem_mem_used{} 25.059381019344624 1635952533657\n# HELP system_mem_used_bytes [GAUGE] Used memory in bytes\n# TYPE system_mem_used_bytes GAUGE\nsystem_mem_used_bytes{} 2.60579328e+08 1635952533657\n# HELP system_mem_total_bytes [GAUGE] Total memory in bytes\n# TYPE system_mem_total_bytes

```
GAUGE\nsystem_mem_total_bytes{} 1.039847424e+09 1635952533657\n# HELP
system_swap_used [GAUGE] Percent of swap used\n# TYPE system_swap_used
GAUGE\nsystem_swap_used{} 0.0736237976528123 1635952533657\n# HELP
system_swap_used_bytes [GAUGE] Used swap in bytes\n# TYPE system_swap_used_bytes
GAUGE\nsystem_swap_used_bytes{} 2.60579328e+08 1635952533657\n# HELP
system_swap_total_bytes [GAUGE] Total swap in bytes\n# TYPE system_swap_total_bytes
GAUGE\nsystem_swap_total_bytes{} 2.147479552e+09 1635952533657\n# HELP
system_load_load1 [GAUGE] System load averaged over 1 minute, high load value
dependant on number of cpus in system\n# TYPE system_load_load1
GAUGE\nsystem_load_load1{} 0.01 1635952533657\n# HELP system_load_load5 [GAUGE]
System load averaged over 5 minute, high load value dependent on number of cpus in
system\n# TYPE system_load_load5 GAUGE\nsystem_load_load5{} 0.02 1635952533657\n#
HELP system_load_load15 [GAUGE] System load averaged over 15 minute, high load value
dependent on number of cpus in system\n# TYPE system_load_load15
GAUGE\nsystem_load_load15{} 0.05 1635952533657\n# HELP system_load_load1_per_cpu
[GAUGE] System load averaged over 1 minute normalized by cpu count, values \\u003e 1
means system may be overloaded\n# TYPE system_load_load1_per_cpu
GAUGE\nsystem_load_load1_per_cpu{} 0.01 1635952533657\n# HELP
system_load_load5_per_cpu [GAUGE] System load averaged over 5 minute normalized by
cpu count, values \\u003e 1 means system may be overloaded\n# TYPE
system_load_load5_per_cpu GAUGE\nsystem_load_load5_per_cpu{} 0.02 1635952533657\n#
HELP system_load_load15_per_cpu [GAUGE] System load averaged over 15 minute
normalized by cpu count, values \\u003e 1 means system may be overloaded\n# TYPE
system_load_load15_per_cpu GAUGE\nsystem_load_load15_per_cpu{} 0.05 1635952533657\n#
HELP system_host_uptime [COUNTER] Host uptime in seconds\n# TYPE system_host_uptime
COUNTER\nsystem_host_uptime{} 10642 1635952533657\n# HELP system_host_processes
[GAUGE] Number of host processes\n# TYPE system_host_processes
GAUGE\nsystem_host_processes{} 116 1635952533657\n\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1635952533,
    "occurrences": 6,
    "occurrences_watermark": 6,
    "output_metric_format": "prometheus_text",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "system-check",
      "namespace": "default",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
```

```
    }
  },
  "secrets": null,
  "is_silenced": false,
  "scheduler": "memory",
  "processed_by": "sensu-centos",
  "pipelines": [],
  "output_metric_thresholds": [
    {
      "name": "system_mem_used",
      "tags": null,
      "null_status": 1,
      "thresholds": [
        {
          "min": "",
          "max": "75.0",
          "status": 1
        },
        {
          "min": "",
          "max": "90.0",
          "status": 2
        }
      ]
    },
    {
      "name": "system_host_processes",
      "tags": [
        {
          "name": "namespace",
          "value": "production"
        }
      ],
      "null_status": 1,
      "thresholds": [
        {
          "min": "5",
          "max": "50",
          "status": 1
        },
        {
          "min": "2",
```

```
            "max": "75",
            "status": 2
          }
        ]
      }
    ]
  }
}
```

Example status and metrics event

The following example resource definition for a status and metrics event contains *both* a check and metrics, retrieved with sensuctl event info:

YML

```
---
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
    command: http-check --url http://localhost && http-perf --url http://localhost
      --warning 1s --critical 2s
    duration: 0.022274319
    env_vars: null
    executed: 1635959379
    handlers: null
    high_flap_threshold: 0
    history:
    - executed: 1635952820
      status: 0
    - executed: 1635952835
      status: 0
    - executed: 1635952850
      status: 0
    - executed: 1635952865
      status: 0
```

```
    - executed: 1635952880
      status: 0
    interval: 5
    is_silenced: false
    issued: 1635952880
    last_ok: 1635952880
    low_flap_threshold: 0
    metadata:
      name: collect-metrics
      namespace: default
    occurrences: 5
    occurrences_watermark: 5
    output: |
      http-check OK: HTTP Status 200 for http://localhost
      http-perf OK: 0.001150s | dns_duration=0.000257,
tls_handshake_duration=0.000000, connect_duration=0.000088,
first_byte_duration=0.001131, total_request_duration=0.001150
    output_metric_format: nagios_perfdata
    output_metric_handlers: null
    pipelines: []
    processed_by: sensu-centos
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
    - http-checks
    scheduler: memory
    secrets: null
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - webserver
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1635959379
```

```
  metadata:
    created_by: admin
    name: sensu-centos
    namespace: default
  redact:
  - password
  - passwd
  - pass
  - api_key
  - api_token
  - access_key
  - secret_key
  - private_key
  - secret
  sensu_agent_version: 6.5.4
  subscriptions:
  - system
  - entity:sensu-centos
  - webserver
  system:
    arch: amd64
    cloud_provider: ""
    hostname: sensu-centos
    libc_type: glibc
    network:
      interfaces:
      - addresses:
        - 127.0.0.1/8
        - ::1/128
        name: lo
      - addresses:
        - 10.0.2.15/24
        - fe80::20b8:8cea:fa4:2e57/64
        mac: 08:00:27:8b:c9:3f
        name: eth0
      - addresses:
        - 192.168.200.95/24
        - fe80::a00:27ff:fe40:ab31/64
        mac: 08:00:27:40:ab:31
        name: eth1
    os: linux
    platform: centos
```

```
      platform_family: rhel
      platform_version: 7.9.2009
      processes: null
      vm_role: guest
      vm_system: vbox
    user: agent
  id: 12545deb-0e0f-480f-addf-34545d5a01c6
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: status_and_metrics_pipeline
  sequence: 5
  timestamp: 1635952880
```

JSON

```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "http-check --url http://localhost && http-perf --url
http://localhost --warning 1s --critical 2s",
      "duration": 0.022274319,
      "env_vars": null,
      "executed": 1635959379,
      "handlers": null,
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1635952820,
          "status": 0
        },
        {
          "executed": 1635952835,
          "status": 0
        },
        {
```

```
          "executed": 1635952850,
          "status": 0
        },
        {
          "executed": 1635952865,
          "status": 0
        },
        {
          "executed": 1635952880,
          "status": 0
        }
      ],
      "interval": 5,
      "is_silenced": false,
      "issued": 1635952880,
      "last_ok": 1635952880,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "collect-metrics",
        "namespace": "default"
      },
      "occurrences": 5,
      "occurrences_watermark": 5,
      "output": "http-check OK: HTTP Status 200 for http://localhost\nhttp-perf OK:
0.001150s | dns_duration=0.000257, tls_handshake_duration=0.000000,
connect_duration=0.000088, first_byte_duration=0.001131,
total_request_duration=0.001150\n",
      "output_metric_format": "nagios_perfdata",
      "output_metric_handlers": null,
      "pipelines": [],
      "processed_by": "sensu-centos",
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "http-checks"
      ],
      "scheduler": "memory",
      "secrets": null,
      "state": "passing",
      "status": 0,
      "stdin": false,
```

```
    "subdue": null,
    "subscriptions": [
      "webserver"
    ],
    "timeout": 0,
    "total_state_change": 0,
    "ttl": 0
  },
  "entity": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "agent",
    "last_seen": 1635959379,
    "metadata": {
      "created_by": "admin",
      "name": "sensu-centos",
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "sensu_agent_version": "6.5.4",
    "subscriptions": [
      "system",
      "entity:sensu-centos",
      "webserver"
    ],
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "sensu-centos",
      "libc_type": "glibc",
      "network": {
        "interfaces": [
```

```
            {
              "addresses": [
                "127.0.0.1/8",
                ":1/128"
              ],
              "name": "lo"
            },
            {
              "addresses": [
                "10.0.2.15/24",
                "fe80::20b8:8cea:fa4:2e57/64"
              ],
              "mac": "08:00:27:8b:c9:3f",
              "name": "eth0"
            },
            {
              "addresses": [
                "192.168.200.95/24",
                "fe80::a00:27ff:fe40:ab31/64"
              ],
              "mac": "08:00:27:40:ab:31",
              "name": "eth1"
            }
          ]
        },
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.9.2009",
        "processes": null,
        "vm_role": "guest",
        "vm_system": "vbox"
      },
      "user": "agent"
    },
    "id": "12545deb-0e0f-480f-addf-34545d5a01c6",
    "pipelines": [
      {
        "type": "Pipeline",
        "api_version": "core/v2",
        "name": "status_and_metrics_pipeline"
      }
```

```
    ],
    "sequence": 5,
    "timestamp": 1635952880
  }
}
```

Metrics data points are not included in events retrieved with sensuctl event info — those events include check output text rather than a set of metrics points. To view metrics points data as shown in the following example, create a pipeline workflow that includes a debug handler that prints events to a JSON file:

```
{
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.9.2009",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::20b8:8cea:fa4:2e57/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:40:ab:31",
```

```
            "addresses": [
              "192.168.200.95/24",
              "fe80::a00:27ff:fe40:ab31/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:sensu-centos",
      "webserver"
    ],
    "last_seen": 1635952880,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin"
    },
    "sensu_agent_version": "6.5.4"
  },
  "check": {
```

```
    "command": "http-check --url http://localhost \\u0026\\u0026 http-perf --url
http://localhost --warning 1s --critical 2s",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "http-checks"
    ],
    "subscriptions": [
      "webserver"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 0.018747388,
    "executed": 1635952880,
    "history": [
      {
        "status": 0,
        "executed": 1635952820
      },
      {
        "status": 0,
        "executed": 1635952835
      },
      {
        "status": 0,
        "executed": 1635952850
      },
      {
        "status": 0,
        "executed": 1635952865
      },
      {
        "status": 0,
        "executed": 1635952880
```

```
      }
    ],
    "issued": 1635952880,
    "output": "http-check OK: HTTP Status 200 for http://localhost\nhttp-perf OK:
0.001059s | dns_duration=0.000235, tls_handshake_duration=0.000000,
connect_duration=0.000083, first_byte_duration=0.001040,
total_request_duration=0.001059\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1635952880,
    "occurrences": 5,
    "occurrences_watermark": 5,
    "output_metric_format": "nagios_perfdata",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "collect-metrics",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "scheduler": "memory",
    "processed_by": "sensu-centos",
    "pipelines": []
  },
  "metrics": {
    "points": [
      {
        "name": "dns_duration",
        "value": 0.000235,
        "timestamp": 1635952880,
        "tags": null
      },
      {
        "name": "tls_handshake_duration",
        "value": 0,
        "timestamp": 1635952880,
        "tags": null
      },
      {
        "name": "connect_duration",
```

```
          "value": 0.000083,
          "timestamp": 1635952880,
          "tags": null
        },
        {
          "name": "first_byte_duration",
          "value": 0.00104,
          "timestamp": 1635952880,
          "tags": null
        },
        {
          "name": "total_request_duration",
          "value": 0.001059,
          "timestamp": 1635952880,
          "tags": null
        }
      ]
    },
    "metadata": {
      "namespace": "default"
    },
    "id": "7cde3e3f-beee-408f-b89a-1edccd0d3edb",
    "sequence": 5,
    "pipelines": [
      {
        "type": "Pipeline",
        "api_version": "core/v2",
        "name": "debug_pipeline"
      }
    ],
    "timestamp": 1635952880
}
```


Create events using the Sensu agent

The Sensu agent is a powerful event producer and monitoring automation tool. You can use Sensu agents to produce events automatically using service checks and metric checks. Sensu agents can also act as a collector for metrics throughout your infrastructure.

- Create events using service checks

- Create events using metric checks
- Create events using the agent API
- Create events using the agent TCP and UDP sockets
- Create events using the StatsD listener

Create events with the core/v2/events API endpoints

You can send events directly to the Sensu observability pipeline using the core/v2 API events endpoint. To create an event, send a JSON event definition with a PUT request to core/v2/events.

If you use the core/v2/events API to create a new event referencing an entity that does not already exist, the sensu-backend will automatically create a proxy entity in the same namespace when the event is published.

> ***NOTE**: An agent cannot belong to, execute checks in, or create events in more than one namespace.*

Manage events

You can manage events using the Sensu web UI, core/v2/events API endpoints, and sensuctl command line tool.

View events

To list all events:

```
sensuctl event list
```

To show event details in the default output format (tabular):

```
sensuctl event info <entity-name> <check-name>
```

***NOTE**: Metrics data points are not included in events retrieved with `sensuctl event info` — these events include check output text rather than a set of metrics points.*

With both the `list` and `info` commands, you can specify an output format using the `--format` flag:

- `yaml` or `wrapped-json` formats for use with `sensuctl create`
- `json` format for use with core/v2/events API endpoints

SHELL

```
sensuctl event info entity-name check-name --format yaml
```

SHELL

```
sensuctl event info entity-name check-name --format wrapped-json
```

SHELL

```
sensuctl event info entity-name check-name --format json
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Delete events

To delete an event:

```
sensuctl event delete entity-name check-name
```

You can use the `--skip-confirm` flag to skip the confirmation step:

```
sensuctl event delete entity-name check-name --skip-confirm
```

You should receive a confirmation message upon success:

```
Deleted
```

Resolve events

You can use sensuctl to change the status of an event to `0` (OK). Events resolved by sensuctl include the output message `Resolved manually by sensuctl`.

```
sensuctl event resolve entity-name check-name
```

You should receive a confirmation message upon success:

```
Resolved
```

Use event data

Observability data in events is a powerful tool for automating monitoring workflows. For example, the `state` attribute provides handlers with a high-level description of check status. Filtering events based on this attribute can help reduce alert fatigue.

State attribute

The `state` event attribute adds meaning to the check status:

- `passing` means the check status is `0` (OK).
- `failing` means the check status is non-zero (WARNING or CRITICAL).
- `flapping` indicates an unsteady state in which the check result status (determined based on per-check high flap threshold and low flap threshold attributes) is not settling on `passing` or `failing` according to the flap detection algorithm.

Flapping typically indicates intermittent problems with an entity, provided your low and high flap threshold settings are properly configured. Although some teams choose to filter out flapping events to reduce unactionable alerts, we suggest sending flapping events to a designated handler for later review. If you repeatedly observe events in flapping state, Sensu's per-check flap threshold configuration allows you to adjust the sensitivity of the flap detection algorithm.

Flap detection algorithm

Sensu uses the same flap detection algorithm as Nagios. Every time you run a check, Sensu records whether the `status` value changed since the previous check. Sensu stores the last 21 `status` values and uses them to calculate the percent state change for the entity/check pair. Then, Sensu's algorithm applies a weight to these status changes: more recent changes have more value than older changes.

After calculating the weighted total percent state change, Sensu compares it with the high flap threshold and low flap threshold set in the check attributes.

- If the entity was **not** already flapping and the weighted total percent state change for the entity/check pair is greater than or equal to the `high_flap_threshold` setting, the entity has started flapping.
- If the entity **was** already flapping and the weighted total percent state change for the entity/check pair is less than the `low_flap_threshold` setting, the entity has stopped flapping.

Depending on the result of this comparison, Sensu will trigger the appropriate event filters based on check attributes like `event.check.high_flap_threshold` and `event.check.low_flap_threshold`.

Occurrences and occurrences watermark

The `occurrences` and `occurrences_watermark` event attributes give you context about recent events for a given entity and check. You can use these attributes within event filters to fine-tune incident notifications and reduce alert fatigue.

Starting at `1`, the `occurrences` attribute increments for events with the same status as the preceding event (OK, WARNING, CRITICAL, or UNKNOWN) and resets whenever the status changes. You can use the `occurrences` attribute to create a state-change-only filter or an interval filter.

The `occurrences_watermark` attribute gives you useful information when looking at events that change status between non-OK (WARNING, CRITICAL, or UNKNOWN) and OK. For these resolution events, the `occurrences_watermark` attribute tells you the number of preceding events with a non-OK status. Sensu resets `occurrences_watermark` to `1` on the first non-OK event. Within a sequence of only OK or only non-OK events, Sensu increments `occurrences_watermark` when the

`occurrences` attribute is greater than the preceding `occurrences_watermark`.

The following table shows the occurrences attributes for a series of example events:

event sequence	`occurrences`	`occurrences_watermark`
1. OK event	`occurrences: 1`	`occurrences_watermark: 1`
2. OK event	`occurrences: 2`	`occurrences_watermark: 2`
3. WARNING event	`occurrences: 1`	`occurrences_watermark: 1`
4. WARNING event	`occurrences: 2`	`occurrences_watermark: 2`
5. WARNING event	`occurrences: 3`	`occurrences_watermark: 3`
6. CRITICAL event	`occurrences: 1`	`occurrences_watermark: 3`
7. CRITICAL event	`occurrences: 2`	`occurrences_watermark: 3`
8. CRITICAL event	`occurrences: 3`	`occurrences_watermark: 3`
9. CRITICAL event	`occurrences: 4`	`occurrences_watermark: 4`
10. OK event	`occurrences: 1`	`occurrences_watermark: 4`
11. CRITICAL event	`occurrences: 1`	`occurrences_watermark: 1`

Event specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For events in this version of Sensu, `api_version` should always be `core/v2`.
required	Required for events in `wrapped-json` or `yaml` format for use with

	`sensuctl create` .
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata

description	Top-level scope that contains the event `namespace` and `created_by` field. The `metadata` map is always at the top level of the check definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Review the metadata attributes for details.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs
example	**YML** `metadata: namespace: default created_by: admin` **JSON** `{ "metadata": { "namespace": "default", "created_by": "admin" } }`

```
}
```

pipelines

description	Name, type, and API version for the pipelines used to process the observability event. Sensu automatically populates the pipelines attributes when the event is created or updated. Read pipelines attributes for more information.
required	false
type	Array

example

YML

```
pipelines:
- type: Pipeline
  api_version: core/v2
  name: incident_alerts
```

JSON

```
{
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "incident_alerts"
    }
  ]
}
```

spec

description	Top-level map that includes the event spec attributes.

required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs **YML**
example	

```yaml
spec:
  check:
    check_hooks:
    command: metrics-curl -u "http://localhost"
    duration: 0.060790838
    env_vars:
    executed: 1552506033
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1552505833
      status: 0
    - executed: 1552505843
      status: 0
    interval: 10
    is_silenced: true
    processed_by: sensu-go-sandbox
    issued: 1552506033
    last_ok: 1552506033
    low_flap_threshold: 0
    metadata:
      name: curl_timings
      namespace: default
    occurrences: 1
    occurrences_watermark: 1
    silenced:
    - webserver:*
    output: |-
      sensu-go.curl_timings.time_total 0.005 1552506033
      sensu-go.curl_timings.time_namelookup 0.004
    output_metric_format: graphite_plaintext
    proxy_entity_name: ''
    publish: true
    round_robin: false
    runtime_assets: []
    state: passing
```

```
    status: 0
    stdin: false
    subdue:
    subscriptions:
    - entity:sensu-go-testing
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1552495139
    metadata:
      name: sensu-go-testing
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
    subscriptions:
    - entity:sensu-go-testing
    system:
      arch: amd64
      hostname: sensu-go-testing
      network:
        interfaces:
        - addresses:
          - 127.0.0.1/8
          - "::1/128"
          name: lo
        - addresses:
          - 10.0.2.15/24
          - fe80::5a94:f67a:1bfc:a579/64
          mac: '08:00:27:8b:c9:3f'
          name: eth0
```

```
      os: linux
      platform: centos
      platform_family: rhel
      platform_version: 7.5.1804
      processes:
    user: agent
  metrics:
    points:
    - name: sensu-go.curl_timings.time_total
      tags: []
      timestamp: 1552506033
      value: 0.005
    - name: sensu-go.curl_timings.time_namelookup
      tags: []
      timestamp: 1552506033
      value: 0.004
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: status_and_metrics_pipeline
  timestamp: 1552506033
  id: 431a0085-96da-4521-863f-c38b480701e9
  sequence: 1
```

JSON

```
{
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "metrics-curl -u \"http://localhost\"",
      "duration": 0.060790838,
      "env_vars": null,
      "executed": 1552506033,
      "handlers": [],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1552505833,
          "status": 0
        },
        {
```

```
          "executed": 1552505843,
          "status": 0
        }
      ],
      "interval": 10,
      "is_silenced": true,
      "processed_by": "sensu-go-sandbox",
      "issued": 1552506033,
      "last_ok": 1552506033,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "curl_timings",
        "namespace": "default"
      },
      "occurrences": 1,
      "occurrences_watermark": 1,
      "silenced": [
        "webserver:*"
      ],
      "output": "sensu-go.curl_timings.time_total 0.005
1552506033\nsensu-go.curl_timings.time_namelookup 0.004",
      "output_metric_format": "graphite_plaintext",
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [],
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "entity:sensu-go-testing"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
    },
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1552495139,
```

```
    "metadata": {
      "name": "sensu-go-testing",
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "subscriptions": [
      "entity:sensu-go-testing"
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-go-testing",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::5a94:f67a:1bfc:a579/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          }
        ]
      },
      "os": "linux",
      "platform": "centos",
```

```
          "platform_family": "rhel",
          "platform_version": "7.5.1804",
          "processes": null
        },
        "user": "agent"
      },
      "metrics": {
        "points": [
          {
            "name": "sensu-go.curl_timings.time_total",
            "tags": [],
            "timestamp": 1552506033,
            "value": 0.005
          },
          {
            "name": "sensu-go.curl_timings.time_namelookup",
            "tags": [],
            "timestamp": 1552506033,
            "value": 0.004
          }
        ]
      },
      "pipelines": [
        {
          "type": "Pipeline",
          "api_version": "core/v2",
          "name": "status_and_metrics_pipeline"
        }
      ],
      "timestamp": 1552506033,
      "id": "431a0085-96da-4521-863f-c38b480701e9",
      "sequence": 1
    }
}
```


type

description

Top-level attribute that specifies the `sensuctl create` resource type.

	Events should always be type `Event`.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: Event` **JSON** `{ "type": "Event" }`

Metadata attributes

created_by	
description	Username of the Sensu user who created the event or last updated the event. Sensu automatically populates the `created_by` field when the event is created or updated.
required	false
type	String
example	**YML** `created_by: "admin"` **JSON** `{ "created_by": "admin" }`

namespace	
description	Sensu RBAC namespace that the event belongs to.
required	false
type	String
default	`default`
example	YML ``` namespace: production ``` JSON ``` { "namespace": "production" } ```

Pipelines attributes

api_version	
description	The Sensu API group and version for the pipeline. Sensu automatically populates the pipelines api_version field when the event is created or updated. For pipelines in this version of Sensu, the api_version is `core/v2`.
required	false
type	String
default	`null`
example	YML

```
api_version: core/v2
```

JSON

```
{
  "api_version": "core/v2"
}
```

name

description	Name of the Sensu pipeline used to process the observability event. Sensu automatically populates the pipeline name field when the event is created or updated.
required	false
type	String
default	`null`
example	**YML** `name: is_incident` **JSON** `{ "name": "is_incident" }`

type

description	The `sensuctl create` resource type for the pipeline. Sensu automatically populates the pipelines type field when the event is created or updated. Pipeline resources are always type `Pipeline`.

required	false
type	String
default	`null`
example	**YML** `type: Pipeline` **JSON** `{ "type": "Pipeline" }`

Spec attributes

check	
description	Check definition used to create the event and information about the status and history of the event. The check scope includes attributes described in the event specification and the check specification.
type	Map
required	true
example	**YML** `check:` `  check_hooks:` `  command: metrics-curl -u "http://localhost"` `  duration: 0.060790838` `  env_vars:` `  executed: 1552506033` `  handlers: []` `  high_flap_threshold: 0` `  history:` `  - executed: 1552505833`

```
    status: 0
  - executed: 1552505843
    status: 0
  interval: 10
  is_silenced: true
  processed_by: sensu-go-sandbox
  issued: 1552506033
  last_ok: 1552506033
  low_flap_threshold: 0
  metadata:
    name: curl_timings
    namespace: default
  occurrences: 1
  occurrences_watermark: 1
  silenced:
  - webserver:*
  output: sensu-go.curl_timings.time_total 0.005
  output_metric_format: graphite_plaintext
  proxy_entity_name: ''
  publish: true
  round_robin: false
  runtime_assets: []
  state: passing
  status: 0
  stdin: false
  subdue:
  subscriptions:
  - entity:sensu-go-testing
  timeout: 0
  total_state_change: 0
  ttl: 0
```

JSON

```
{
  "check": {
    "check_hooks": null,
    "command": "metrics-curl -u \"http://localhost\"",
    "duration": 0.060790838,
    "env_vars": null,
    "executed": 1552506033,
```

```
    "handlers": [],
    "high_flap_threshold": 0,
    "history": [
      {
        "executed": 1552505833,
        "status": 0
      },
      {
        "executed": 1552505843,
        "status": 0
      }
    ],
    "interval": 10,
    "is_silenced": true,
    "processed_by": "sensu-go-sandbox",
    "issued": 1552506033,
    "last_ok": 1552506033,
    "low_flap_threshold": 0,
    "metadata": {
      "name": "curl_timings",
      "namespace": "default"
    },
    "occurrences": 1,
    "occurrences_watermark": 1,
    "silenced": [
      "webserver:*"
    ],
    "output": "sensu-go.curl_timings.time_total 0.005",
    "output_metric_format": "graphite_plaintext",
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [],
    "state": "passing",
    "status": 0,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "entity:sensu-go-testing"
    ],
    "timeout": 0,
    "total_state_change": 0,
```

```
    "ttl": 0
  }
}
```

entity	
description	Entity attributes from the originating entity (agent or proxy). For events created with the core/v2/events API, if the event's entity does not already exist, the sensu-backend automatically creates a proxy entity when the event is published.
type	Map
required	true
example	YML entity: deregister: **false** deregistration: {} entity_class: agent last_seen: 1552495139 metadata: name: sensu-go-testing namespace: default redact: - password - passwd - pass - api_key - api_token - access_key - secret_key - private_key - secret subscriptions: - entity:sensu-go-testing system: arch: amd64

```
    hostname: sensu-go-testing
    network:
      interfaces:
      - addresses:
        - 127.0.0.1/8
        - "::1/128"
        name: lo
      - addresses:
        - 10.0.2.15/24
        - fe80::5a94:f67a:1bfc:a579/64
        mac: '08:00:27:8b:c9:3f'
        name: eth0
    os: linux
    platform: centos
    platform_family: rhel
    platform_version: 7.5.1804
  user: agent
```

JSON

```
{
  "entity": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "agent",
    "last_seen": 1552495139,
    "metadata": {
      "name": "sensu-go-testing",
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "subscriptions": [
```

```
      "entity:sensu-go-testing"
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-go-testing",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::5a94:f67a:1bfc:a579/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          }
        ]
      },
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804"
    },
    "user": "agent"
  }
}
```


id	
description	Universally unique identifier (UUID) for the event. Logged as `event_id`. Sensu automatically populates the `id` value for the event.

required	false
type	String

YML

example	<pre>id: 431a0085-96da-4521-863f-c38b480701e9</pre>

JSON

<pre>{
  "id": "431a0085-96da-4521-863f-c38b480701e9"
}</pre>

metrics

description	Metrics collected by the entity in Sensu metric format. Review the metrics attributes.
type	Map
required	false

YML

example	<pre>metrics: points: - name: sensu-go.curl_timings.time_total tags: [] timestamp: 1552506033 value: 0.005 - name: sensu-go.curl_timings.time_namelookup tags: [] timestamp: 1552506033 value: 0.004</pre>

JSON

<pre>{
  "metrics": {
</pre>

```
    "points": [
      {
        "name": "sensu-go.curl_timings.time_total",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "sensu-go.curl_timings.time_namelookup",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.004
      }
    ]
  }
}
```

sequence

description	Event sequence number. The Sensu agent sets the sequence to 1 at startup, then increments the sequence by 1 for every successive check execution or keepalive event. If the agent restarts or reconnects to another backend, the sequence value resets to 1. A sequence value of 0 indicates that an outdated or non-conforming agent generated the event. Sensu only increments the sequence for agent-executed events. Sensu does not update the sequence for events created with the core/v2/events API.
required	false
type	Integer

YML

example

```
sequence: 1
```

JSON

```
{
  "sequence": 1
}
```

timestamp	
description	Time that the event occurred. In seconds since the Unix epoch. Sensu automatically populates the timestamp value for the event. For events created with the core/v2/events API, you can specify a `timestamp` value in the request body.
required	false
type	Integer
default	Time that the event occurred
example	**YML** `timestamp: 1522099512` **JSON** `{ "timestamp": 1522099512 }`

Check attributes

Sensu events include a `check` scope that contains information about how the event was created, including any attributes defined in the check specification, and information about the event and its history, including the attributes defined below.

duration

description	Command execution time. In seconds.
required	false
type	Float

YML

example

```
duration: 1.903135228
```

JSON

```
{
  "duration": 1.903135228
}
```

executed

description	Time at which the check request was executed. In seconds since the Unix epoch. The difference between a request's `issued` and `executed` values is the request latency. For agent-executed checks, Sensu automatically populates the `executed` value. For events created with the core/v2/events API, the default `executed` value is `0` unless you specify a value in the request body.
required	false
type	Integer

YML

example

```
executed: 1522100915
```

JSON

```
{
  "executed": 1522100915
}
```

history	
description	Check status history for the last 21 check executions. Read history attributes. Sensu automatically populates the history attributes with check execution data. To store more than the last 21 check executions, use one of our long-term event storage integrations.
required	false
type	Array

YML

example

```
history:
- executed: 1552505983
  status: 0
- executed: 1552505993
  status: 0
```

JSON

```
{
  "history": [
    {
      "executed": 1552505983,
      "status": 0
    },
    {
      "executed": 1552505993,
      "status": 0
    }
```

```
    ]
}
```

is_silenced

description	If `true`, the event was silenced at the time of processing. Otherwise, `false`. If `true`, the event. Check definitions also list the silenced entries that match the event in the `silenced` array.
required	false
type	Boolean
example	**YML** `is_silenced: true` **JSON** `{ "is_silenced": "true" }`

issued

description	Time that the check request was issued. In seconds since the Unix epoch. The difference between a request's `issued` and `executed` values is the request latency. For agent-executed checks, Sensu automatically populates the `issued` value. For events created with the core/v2/events API, the default `issued` value is `0` unless you specify a value in the request body.
required	false

type	Integer

YML

example	`issued: 1552506033`

JSON

```
{
  "issued": 1552506033
}
```

last_ok

description	Last time that the check returned an OK status (`0`). In seconds since the Unix epoch. For agent-executed checks, Sensu automatically populates the `last_ok` value. For events created with the core/v2/events API, the `last_ok` attribute will default to `0` even if you specify OK status in the request body.
required	false
type	Integer

YML

example	`last_ok: 1552506033`

JSON

```
{
  "last_ok": 1552506033
}
```

occurrences

description	Number of preceding events with the same status as the current event (OK, WARNING, CRITICAL, or UNKNOWN). Starting at `1`, the `occurrences` attribute increments for events with the same status as the preceding event and resets whenever the status changes. Read Use event data for more information. Sensu automatically populates the `occurrences` value. For events created with the core/v2/events API, Sensu overwrites any `occurences` value you specify in the request body with the correct value.
required	false
type	Integer greater than 0
example	**YML** `occurrences: 1` **JSON** `{ "occurrences": 1 }`

occurrences_watermark

description	For incident and resolution events, the number of preceding events with an OK status (for incident events) or non-OK status (for resolution events). The `occurrences_watermark` attribute gives you useful information when looking at events that change status between OK (`0`)and non-OK (`1` -WARNING, `2` -CRITICAL, or UNKNOWN). Sensu resets `occurrences_watermark` to `1` whenever an event for a given entity and check transitions between OK and non-OK. Within a sequence of only OK or only non-OK events, Sensu increments `occurrences_watermark` only when the `occurrences` attribute is greater than the preceding

	`occurrences_watermark`. Read Use event data for more information. Sensu automatically populates the `occurrences_watermark` value. For events created with the core/v2/events API, Sensu overwrites any `occurences_watermark` value you specify in the request body with the correct value.
required	false
type	Integer greater than 0
example	**YML** `occurrences_watermark: 1` **JSON** `{ "occurrences_watermark": 1 }`

output

description	Output from the execution of the check command.
required	false
type	String
example	**YML** `output: "sensu-go.curl_timings.time_total 0.005` **JSON** `{ "output": "sensu-go.curl_timings.time_total 0.005" }`

processed_by

description	The name of the agent that processed the event. Useful for determining which agent processed an event executed by a proxy check request or a POST request to the events API.
required	false
type	String

YML

example

```
processed_by: sensu-go-sandbox
```

JSON

```
{
  "processed_by": "sensu-go-sandbox"
}
```

silenced

description	Array of silencing entries that match the event. The `silenced` attribute is only present for events if one or more silencing entries matched the event at time of processing. If the `silenced` attribute is not present in an event, the event was not silenced at the time of processing.
required	false
type	Array

YML

example

```
silenced:
- webserver:*
```

JSON

```
{
  "silenced": [
    "webserver:*"
  ]
}
```

state

description	State of the check: `passing` (status `0`), `failing` (status other than `0`), or `flapping`. Use the `low_flap_threshold` and `high_flap_threshold` check attributes to configure `flapping` state detection. Sensu automatically populates the `state` based on the `status`.
required	false
type	String

YML

```
state: passing
```

JSON

```
{
  "state": "passing"
}
```

status

description	Exit status code produced by the check. - `0` indicates OK - `1` indicates WARNING

- `2` indicates CRITICAL

Exit status codes other than `0`, `1`, or `2` indicate an UNKNOWN or custom status.

For agent-executed checks, Sensu automatically populates the `status` value based on the check result. For events created with the core/v2/events API, Sensu assumes the status is `0` (OK) unless you specify a non-zero value in the request body.

required	false
type	Integer

YML

```
status: 0
```

JSON

```
{
  "status": 0
}
```

total_state_change

description	Total state change percentage for the check's history. For agent-executed checks, Sensu automatically populates the `total_state_change` value. For events created with the core/v2/events API, the `total_state_change` attribute will default to `0` even if you specify a different value or change the `status` value in the request body.
required	false
type	Integer

YML

example	`total_state_change: 0`

JSON

```
{
  "total_state_change": 0
}
```

History attributes

executed	
description	Time at which the check request was executed. In seconds since the Unix epoch. Sensu automatically populates the `executed` value with check execution data. For events created with the core/v2/events API, the `executed` default value is `0`.
required	false
type	Integer
example	**YML** `executed: 1522100915` **JSON** `{ "executed": 1522100915 }`

status	
description	Exit status code produced by the check.

- `0` indicates OK
- `1` indicates WARNING
- `2` indicates CRITICAL

Exit status codes other than `0`, `1`, or `2` indicate an UNKNOWN or custom status.

Sensu automatically populates the `status` value with check execution data.

required	false
type	Integer
example	**YML** `status: 0` **JSON** `{ "status": 0 }`

Metrics attributes

handlers	
description	Array of Sensu handlers to use for events created by the check. Each array item must be a string.
required	false
type	Array
example	**YML** `handlers:`

```
- influx-db
```

JSON

```
{
  "handlers": [
    "influx-db"
  ]
}
```

points	
description	Metrics data points, including a name, timestamp, value, and tags. Read points attributes.
required	false
type	Array
example	**YML** `points:` `- name: sensu-go.curl_timings.time_total` `  tags:` `  - name: response_time_in_ms` `  value: '101'` `  timestamp: 1552506033` `  value: 0.005` `- name: sensu-go.curl_timings.time_namelookup` `  tags:` `  - name: namelookup_time_in_ms` `  value: '57'` `  timestamp: 1552506033` `  value: 0.004` **JSON** `{` `  "points": [`

```
      {
        "name": "sensu-go.curl_timings.time_total",
        "tags": [
          {
            "name": "response_time_in_ms",
            "value": "101"
          }
        ],
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "sensu-go.curl_timings.time_namelookup",
        "tags": [
          {
            "name": "namelookup_time_in_ms",
            "value": "57"
          }
        ],
        "timestamp": 1552506033,
        "value": 0.004
      }
    ]
  }
```


Points attributes

name	
description	Metric name in the format `$entity.$check.$metric` where `$entity` is the entity name, `$check` is the check name, and `$metric` is the metric name.
required	false
type	String YML
example	

```
name: sensu-go.curl_timings.time_total
```

JSON

```
{
  "name": "sensu-go.curl_timings.time_total"
}
```

tags

description	Optional tags to include with the metric. Each element of the array must be a hash that contains two key-value pairs: the `name` of the tag and the `value`. Both values of the pairs must be strings.
required	false
type	Array

YML

example

```
tags:
- name: response_time_in_ms
  value: '101'
```

JSON

```
{
  "tags": [
    {
      "name": "response_time_in_ms",
      "value": "101"
    }
  ]
}
```

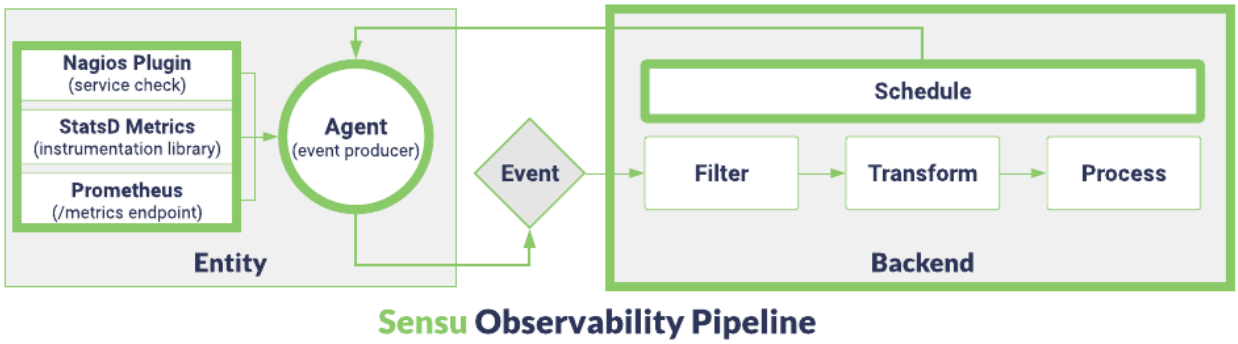
timestamp

description	Time at which the metric was collected. In seconds since the Unix epoch. Sensu automatically populates the timestamp values for metrics data points.
required	false
type	Integer
example	**YML** `timestamp: 1552506033` **JSON** `{ "timestamp": 1552506033 }`

value

description	Metric value.
required	false
type	Float
example	**YML** `value: 0.005` **JSON** `{ "value": 0.005 }`

Schedule observability data collection



Sensu Observability Pipeline

Next **or click any element in the pipeline to jump to it.**

Sensu's schedule function is based on subscriptions: transport topics to which the Sensu backend publishes check requests. The subscriptions you specify in your Sensu agent definition determine which checks the agent will execute. The Sensu backend schedules checks, publishes check execution requests to entities, and processes the observation data (events) it receives from the agent.

Agent and backend

The Sensu agent is a lightweight process that runs on the infrastructure components you want to monitor and observe. The agent registers with the Sensu backend as an entity with `type: "agent"`. Agent entities are responsible for creating status and metrics events to send to the backend event pipeline.

The Sensu backend includes an integrated structure for scheduling checks using subscriptions and an event pipeline that applies event filters, mutators, and handlers, an embedded etcd datastore for storing configuration and state, and the Sensu API, Sensu web UI, and sensuctl command line tool.

The Sensu agent is available for Linux, macOS, and Windows. The Sensu backend is available for Debian- and RHEL-family distributions of Linux. Learn more in the agent and backend references.

Follow the installation guide to install the agent and backend.

Subscriptions

Subscriptions are at the core of Sensu's publish/subscribe pattern of communication: subscriptions are transport topics to which the Sensu backend publishes check requests. Sensu entities become subscribers to these topics via their individual `subscriptions` attribute.

Each Sensu agent's defined set of subscriptions determine which checks the agent will execute. Agent subscriptions allow Sensu to request check executions on a group of systems at a time instead of a traditional 1:1 mapping of configured hosts to monitoring checks.

In each check's definition, you can specify which subscriptions should run the check. At the same time, your entities are "subscribed" to these subscriptions. Subscriptions make sure your entities automatically run the appropriate checks for their functionality.

The following example shows the resource definition for a check with the `system` and `linux` subscriptions. This check would run on any entities whose definitions also specify the `system` or `linux` subscriptions.

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check-cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers:
  - slack
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
```

```
  subscriptions:
  - system
  - linux
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-cpu"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
```

```
}
```

Subscriptions typically correspond to a specific role or responsibility. For example, you might add all the checks you want to run on your database entities to a `database` subscription. Rather than specifying these checks individually for every database you are monitoring, you add the `database` subscription to your database entities and they run the desired checks automatically.

Read the subscriptions reference to learn more.

Communication between the agent and backend

The Sensu agent uses WebSocket (ws) protocol to send and receive JSON messages with the Sensu backend. For optimal network throughput, agents will attempt to negotiate the use of Protobuf serialization when communicating with a Sensu backend that supports it. This communication is via clear text by default.

Follow Secure Sensu to configure the backend and agent for WebSocket Secure (wss) encrypted communication.

Agent reference

Example Sensu agent configuration file (download)

The Sensu agent is a lightweight client that runs on the infrastructure components you want to monitor. Agents register with the Sensu backend as entities with `type: "agent"`. Agent entities are responsible for creating check and metrics events to send to the backend event pipeline.

The Sensu agent is available for Linux, macOS, and Windows. For Windows operating systems, the Sensu agent uses `cmd.exe` for the execution environment. For all other operating systems, the Sensu agent uses the Bourne shell (sh).

Read the installation guide to install the agent.

Agent authentication

The Sensu agent authenticates to the Sensu backend via WebSocket transport by either built-in basic authentication (username and password) or mutual transport layer security (mTLS) authentication.

Username and password authentication

The default mechanism for agent authentication is built-in basic authentication with username and password. The Sensu agent uses username and password authentication unless mTLS authentication has been explicitly configured.

For username and password authentication, sensu-agent joins the username and password with a colon and encodes them as a Base64 value. Sensu provides the encoded string as the value of the `Authorization` HTTP header — for example, `Authorization: Basic YWdlbnQ6UEBzc3cwcmQh` — to authenticate to the Sensu backend.

When using username and password authentication, sensu-agent also sends the following HTTP headers in requests to the backend:

- `Sensu-User`: the username in plaintext
- `Sensu-AgentName`: the agent's configured name in plaintext

- `Sensu-Subscriptions`: the agent's subscriptions in a comma-separated plaintext list
- `Sensu-Namespace`: the agent's configured namespace in plaintext

mTLS authentication

When mTLS is configured for both the Sensu agent and backend, the agent uses mTLS authentication instead of the default username and password authentication.

Sensu backends that are configured for mTLS authentication will no longer accept agent authentication via username and password. Agents that are configured to use mTLS authentication cannot authenticate with the backend unless the backend is configured for mTLS.

To configure the agent and backend for mTLS authentication:

- In the backend configuration, specify valid certificate and key files as values for the `agent-auth-cert-file` and `agent-auth-key-file` parameters (e.g. `backend-1.pem` and `backend-1-key.pem`, respectively).
- In the agent configuration, specify valid certificate and key files as values for the `cert-file` and `key-file` parameters (e.g. `agent.pem` and `agent-key.pem`, respectively).

> ***NOTE**: For detailed information about the certificates and keys required for mTLS authentication, read Generate certificates for your Sensu installation. For information about using the certificates and keys to secure your configuration, read Secure Sensu.*

The agent and backend will compare the provided certificates with the trusted CA certificate either in the system trust store or specified explicitly as the `agent-auth-trusted-ca-file` in the backend configuration and `trusted-ca-file` in the agent configuration.

When using mTLS authentication, sensu-agent sends the following HTTP headers in requests to the backend:

- `Sensu-AgentName`: the agent's configured name in plaintext
- `Sensu-Subscriptions`: the agent's subscriptions in a comma-separated, plaintext list
- `Sensu-Namespace`: the agent's configured namespace in plaintext

If the Sensu agent is configured for mTLS authentication, it will not send the `Authorization` HTTP header.

Certificate bundles or chains

The Sensu agent supports all types of certificate bundles (or chains) as long as the agent (or leaf) certificate is the *first* certificate in the bundle. This is because the Go standard library assumes that the first certificate listed in the PEM file is the leaf certificate — the certificate that the program will use to show its own identity.

If you send the leaf certificate alone instead of sending the whole bundle with the leaf certificate first, you will receive a `certificate not signed by trusted authority` error. You must present the whole chain to the remote so it can determine whether it trusts the presented certificate through the chain.

Certificate revocation check

The Sensu backend checks certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP) endpoints for agent mTLS, etcd client, and etcd peer connections whose remote sides present X.509 certificates that provide CRL and OCSP revocation information.

Communication between the agent and backend

The Sensu agent uses WebSocket (ws) protocol to send and receive JSON messages with the Sensu backend. For optimal network throughput, agents will attempt to negotiate the use of Protobuf serialization when communicating with a Sensu backend that supports it. This communication is via clear text by default.

Follow Secure Sensu to configure the backend and agent for WebSocket Secure (wss) encrypted communication.

> ***NOTE**: For information about agent transport status, use the /health API.*

Connection failure

Although connection failure may be due to socket errors like unexpectedly closed connections and TLS handshake failures, the Sensu agent generally keeps retrying connections to each URL in the `backend-url` list until it is successfully connected to a backend URL or you stop the process.

When you start up a Sensu agent configured with multiple `backend-url` values, the agent shuffles

the `backend-url` list and attempts to connect to the first URL in the shuffled list. If the agent cannot establish a WebSocket connection with the first URL within the number of seconds specified for the `backend-handshake-timeout`, the agent abandons the connection attempt and tries the next URL in the shuffled list.

When the agent establishes a WebSocket connection with a backend URL within the `backend-handshake-timeout` period, the agent sends a heartbeat message to the backend at the specified `backend-heartbeat-interval`. For every heartbeat the agent sends, the agent expects the connected backend to send a heartbeat response within the number of seconds specified for the `backend-heartbeat-timeout`. If the connected backend does not respond within the `backend-heartbeat-timeout` period, the agent closes the connection and attempts to connect to the next backend URL in the shuffled list.

The agent iterates through the shuffled `backend-url` list until it successfully establishes a WebSocket connection with a backend, returning to the first URL if it fails to connect with the last URL in the list.

> ***NOTE**: Sensu's WebSocket connection heartbeat message and keepalive monitoring mechanism are different, although they have similar purposes.*
>
> *The `backend-heartbeat-interval` and `backend-heartbeat-timeout` are specifically configured for the WebSocket connection heartbeat message the agent sends when it connects to a backend URL.*
>
> *Keepalive monitoring is more fluid — it permits agents to reconnect any number of times within the configured timeout. As long as the agent can successfully send one event to any backend within the timeout, the keepalive logic is satisfied.*

Synchronize time between agents and the backend

System clocks between agents and the backend should be synchronized to a central NTP server. If system time is out of sync, it may cause issues with keepalive, metric, and check alerts.

Agent connection to a cluster

Agents can connect to a Sensu cluster by specifying any Sensu backend URL in the cluster in the `backend-url` configuration option.

For more information about clustering, read Backend datastore configuration and Run a Sensu cluster.

Keepalive monitoring

Sensu keepalives are the heartbeat mechanism used to ensure that all registered agents are operational and able to reach the Sensu backend.Sensu agents publish keepalive events containing entity configuration data to the Sensu backend according to the interval specified by the `keepalive-interval` configuration option.All Sensu agent data provided in keepalive events is stored in the agent registry and used to add context to Sensu events and detect Sensu agents in an unhealthy state.

If a Sensu agent fails to send keepalive events over the period specified by the `keepalive-critical-timeout` configuration option, the Sensu backend creates a keepalive **critical** alert in the Sensu web UI.The `keepalive-critical-timeout` is set to `0` (disabled) by default to help ensure that it will not interfere with your `keepalive-warning-timeout` setting.

If a Sensu agent fails to send keepalive events over the period specified by the `keepalive-warning-timeout` configuration option, the Sensu backend creates a keepalive **warning** alert in the Sensu web UI.The value you specify for `keepalive-warning-timeout` must be lower than the value you specify for `keepalive-critical-timeout`.

> ***NOTE**: If you set the `deregister` configuration option to `true`, when a Sensu agent process stops, the Sensu backend will deregister the corresponding entity.*
>
> *Deregistration prevents and clears alerts for failing keepalives for agent entities — the backend does not distinguish between intentional shutdown and failure.As a result, if you set `deregister` to `true` and an agent process stops for any reason, you will not receive alerts for keepalive events in the web UI.*
>
> *If you want to receive alerts for failing keepalives, set the deregister configuration option to `false`.*

You can use keepalives to identify unhealthy systems and network partitions, send notifications, trigger auto-remediation, and automatically register and deregister agent entities, among other useful actions. The agent maps `keepalive-critical-timeout` and `keepalive-warning-timeout` values to certain event check attributes, so you can also create time-based event filters to reduce alert fatigue for agent keepalive events.

> ***NOTE**: Automatic keepalive monitoring is not supported for proxy entities because they cannot run a Sensu agent.Use the core/v2/events API to send manual keepalive events for proxy entities.*

Process keepalive events

Process keepalive events with a pipeline or handler.

Keepalive pipelines

Use the `keepalive-pipelines` configuration option to send keepalive events to any pipeline you have configured.

To specify pipelines for the `keepalive-pipelines` option, use the fully qualified name for pipelines (`core/v2.Pipeline`) plus the pipeline name (e.g. `slack` or `store-keepalives`). For example:

SHELL

```
sensu-agent start --keepalive-pipelines
core/v2.Pipeline.slack,core/v2.Pipeline.store-keepalives
```

YML

```
keepalive-pipelines:
- core/v2.Pipeline.slack
- core/v2.Pipeline.store-keepalives
```

If you do not specify a pipeline with the `keepalive-pipelines` option, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI for keepalives.

Keepalive handlers

You can use a keepalive handler to connect keepalive events to your monitoring workflows. Sensu looks for an event handler named `keepalive` and automatically uses it to process keepalive events.

Suppose you want to receive Slack notifications for keepalive alerts, and you already have a Slack handler set up to process events. To process keepalive events using the Slack handler, create a handler set named `keepalive` and add the `slack` handler to the `handlers` array. The resulting `keepalive` handler set configuration looks like this:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: keepalive
spec:
  handlers:
  - slack
  type: set
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "keepalive"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack"
    ]
  }
}
```

You can also use the `keepalive-handlers` configuration option to send keepalive events to any handler you have configured. If you do not specify a keepalive handler with the `keepalive-handlers` option, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI.

Create observability events using service checks

The Sensu backend coordinates check execution for you by comparing the subscriptions you specify in your checks and entities to determine which entities should receive execution requests for a given check.

Sensu uses the publish/subscribe pattern of communication, which allows automated registration and

deregistration of ephemeral systems. At the core of this model are Sensu subscriptions, which you specify in checks and entities to determine which entities should receive execution requests for a given check. Subscriptions often correspond with the roles and responsibilities assigned to the entity, such as `webserver` or `database`.

Subscriptions determine which checks the agent will execute. For an agent to execute a check, at least one entity must include a subscription that matches a subscription in the check definition. Read the subscriptions reference for more information.

After receiving a check request from the Sensu backend, the agent:

1. Applies any tokens that match attribute values in the check definition.
2. Fetches dynamic runtime assets and stores them in its local cache.

   By default, agents cache dynamic runtime asset data at `/var/cache/sensu/sensu-agent` (Linux) or `C:\ProgramData\sensu\cache\sensu-agent` (Windows). To specify a different cache location, use the `cache-dir` configuration attribute.
3. Executes the check command.
4. Executes any hooks specified by the check based on the exit status.
5. Creates an event that contains information about the applicable entity, check, and metric.

The Sensu backend then processes the event by applying event filters, mutators, and handlers.

Proxy entities

Proxy entities allow Sensu to monitor external resources on systems or devices where a Sensu agent cannot be installed, like a network switch.

The Sensu backend stores proxy entity definitions (unlike agent entities, which the agent stores). When the backend requests a check that includes a `proxy_entity_name`, the agent includes the provided entity information in the observation data in events in place of the agent entity data.

Read the entities reference and Monitor external resources for more information about monitoring proxy entities.

Create observability events using the agent API

The Sensu agent API allows external sources to send monitoring data to Sensu without requiring the external sources to know anything about Sensu's internal implementation. The agent API listens on the address and port specified with the agent API configuration options.

The agent API supports only unsecured HTTP requests (no HTTPS). Requests for unknown endpoints will result in an `HTTP 404 Not Found` response.

`/events` (POST)

The agent API provides HTTP POST access to publish observability events to the Sensu backend via the `/events` endpoint.

The agent places events created via the agent API `/events` endpoint into a queue stored on disk. In case of a loss of connection with the backend or agent shutdown, the agent preserves queued event data. When the connection is reestablished, the agent sends the queued events to the backend.

The agent API `/events` endpoint uses a configurable burst limit and rate limit for relaying events to the backend. Read API configuration to configure the `events-burst-limit` and `events-rate-limit` options.

Example POST request to events endpoint

The following example submits an HTTP POST request to the agent API `/events` endpoint. The request creates an event for a check named `check-mysql-status` with the output `could not connect to mysql` and a status of `1` (warning). The agent responds with an `HTTP 202 Accepted` response to indicate that the event has been added to the queue to be sent to the backend.

In this example, the event will be processed according to an `incident_alerts` pipeline.

> ***NOTE**: For HTTP POST requests to the agent API `/events` endpoint, check-specific spec attributes are not required. If you do want to include spec attributes, list them as individual top-level attributes within the event's `check` scope.*

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
```

```
  "check": {
    "metadata": {
      "name": "check-mysql-status"
    },
    "status": 1,
    "output": "could not connect to mysql"
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ]
}' \
http://127.0.0.1:3031/events
```

***PRO TIP**: To use the agent API `/events` endpoint to create proxy entities, include a `proxy_entity_name` attribute within the `check` scope.*

Detect silent failures

You can use the Sensu agent API in combination with the check time-to-live (TTL) attribute to detect silent failures. This creates what's commonly referred to as a "dead man's switch".

With check TTLs, Sensu can set an expectation that a Sensu agent will publish additional events for a check within the period of time specified by the TTL attribute. If a Sensu agent fails to publish an event before the check TTL expires, the Sensu backend creates an event with a status of `1` (warning) to indicate the expected event was not received. For more information about check TTLs, read the checks reference.

If you use the check TTL attribute along with the Sensu agent API to enable tasks that run outside of Sensu's check scheduling to emit events, these events create a dead man's switch: if the task fails for any reason, the lack of an "all clear" event from the task will notify operators of the silent failure, which might otherwise be missed. If an external source sends an event with a check TTL to the Sensu agent API, Sensu expects another event from the same external source before the TTL expires.

Here's an example of external event input via the Sensu agent API that uses a check TTL to create a dead man's switch for MySQL backups. Assume that a MySQL backup script runs periodically, and you expect the job to take a little less than 7 hours to complete.

- If the job completes successfully, you want a record of it, but you don't need to receive an alert.
- If the job fails or continues running longer than the expected 7 hours, you do need to receive an alert.

The script can send an event that tells the Sensu backend to expect an additional event with the same name within 7 hours of the first event:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "mysql-backup-job"
    },
    "status": 0,
    "output": "mysql backup initiated",
    "ttl": 25200
  }
}' \
http://127.0.0.1:3031/events
```

When the script submitted this initial event to the agent API, you recorded in the Sensu backend that your script started. You also configured the dead man's switch by including the `ttl` attribute, so you'll receive an alert if the job fails or runs for too long. Although it is possible for your script to handle errors gracefully and emit additional observability events, this approach allows you to worry less about handling every possible error case. A lack of additional events before the 7-hour period elapses results in an alert.

If your backup script runs successfully, it can send an additional event without the TTL attribute, which removes the dead man's switch:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "mysql-backup-job"
    },
```

```
    "status": 0,
    "output": "mysql backup ran successfully!"
  }
}' \
http://127.0.0.1:3031/events
```

Omitting the TTL attribute from this event also removes the dead man's switch being monitored by the Sensu backend. This effectively sounds the "all clear" for this iteration of the task.

API specification

/events (POST)	
description	Accepts JSON event data and passes the event to the Sensu backend event pipeline for processing.
example url	http://hostname:3031/events
payload example	`{ "check": { "metadata": { "name": "check-mysql-status" }, "status": 1, "output": "could not connect to mysql" } }`
payload attributes	Required: - `check`: All check data must be within the `check` scope - `metadata`: The `check` scope must contain a `metadata` scope - `name`: The `metadata` scope must contain the `name` attribute with a string that represents the name of the monitoring check Optional:

	¬ Any other attributes supported by the Sensu check specification
response codes	¬ **Success**: 202 (Accepted) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

`/healthz` (GET)

The agent API `/healthz` endpoint provides HTTP GET access to the status of the Sensu agent via the agent API.

Example

In the following example, an HTTP GET request is submitted to the agent API `/healthz` endpoint:

```
curl http://127.0.0.1:3031/healthz
```

The request results in a healthy response:

```
ok
```

API specification

/healthz (GET)	
description	Returns the agent status: - `ok` if the agent is active and connected to a Sensu backend. - `sensu backend unavailable` if the agent cannot connect to a backend.
example url	http://hostname:3031/healthz

Create observability events using the StatsD listener

Sensu agents include a listener to send StatsD metrics to the event pipeline.By default, Sensu agents listen on UDP socket 8125 for messages that follow the StatsD line protocol and send metric events for handling by the Sensu backend.

For example, you can use the Netcat utility to send metrics to the StatsD listener:

```
echo 'abc.def.g:10|c' | nc -w1 -u localhost 8125
```

Sensu does not store metrics received through the StatsD listener, so it's important to configure event handlers.

StatsD line protocol

The Sensu StatsD listener accepts messages that are formatted according to the StatsD line protocol:

```
<metricname>:<value>|<type>
```

For more information about StatsD, read the StatsD documentation.

Configure the StatsD listener

To configure the StatsD listener, specify the `statsd-event-handlers` configuration option in the agent configuration and start the agent.For example, to start an agent that sends StatsD metrics to InfluxDB, run:

```
sensu-agent --statsd-event-handlers influx-db
```

Use the StatsD configuration options to change the default settings for the StatsD listener address, port, and flush interval.For example, to start an agent with a customized address and flush interval, run:

```
sensu-agent --statsd-event-handlers influx-db --statsd-flush-interval 1 --statsd-metrics-host 123.4.5.11 --statsd-metrics-port 8125
```

Create observability events using the agent TCP and UDP sockets

> ***NOTE**: The agent TCP and UDP sockets are deprecated in favor of the agent API.*

Sensu agents listen for external monitoring data using TCP and UDP sockets. The agent sockets accept JSON event data and pass events to the Sensu backend event pipeline for processing. The TCP and UDP sockets listen on the address and port specified by the socket configuration options.

Use the TCP socket

This example demonstrates external monitoring data input via the Sensu agent TCP socket. The example uses Bash's built-in `/dev/tcp` file to communicate with the Sensu agent socket:

```
echo '{"name": "check-mysql-status", "status": 1, "output": "error!"}' > /dev/tcp/localhost/3030
```

You can also use the Netcat utility to send monitoring data to the agent socket:

```
echo '{"name": "check-mysql-status", "status": 1, "output": "error!"}' | nc localhost 3030
```

Use the UDP socket

This example demonstrates external monitoring data input via the Sensu agent UDP socket. The example uses Bash's built-in `/dev/udp` file to communicate with the Sensu agent socket:

```
echo '{"name": "check-mysql-status", "status": 1, "output": "error!"}' >
```

```
/dev/udp/127.0.0.1/3030
```

You can also use the Netcat utility to send monitoring data to the agent socket:

```
echo '{"name": "check-mysql-status", "status": 1, "output": "error!"}' | nc -u -v
127.0.0.1 3030
```

Socket event format

The agent TCP and UDP sockets use a special event data format designed for backward compatibility with Sensu Core 1.x check results. Attributes specified in socket events appear in the resulting event data passed to the Sensu backend.

Example socket input: Minimum required attributes

```
{
  "name": "check-mysql-status",
  "status": 1,
  "output": "error!"
}
```

Example socket input: All attributes

```
{
  "name": "check-http",
  "status": 1,
  "output": "404",
  "source": "sensu-docs-site",
  "executed": 1550013435,
  "duration": 1.903135228,
  "handlers": ["slack", "influxdb"]
}
```

Socket event specification

***NOTE**: The Sensu agent socket ignores any attributes that are not included in this specification.*

name

description	Check name.
required	true
type	String
example	`{ "name": "check-mysql-status" }`

status

description	Check execution exit status code. An exit status code of `0` (zero) indicates `OK`, `1` indicates `WARNING`, and `2` indicates `CRITICAL`. Exit status codes other than `0`, `1`, and `2` indicate an `UNKNOWN` or custom status.
required	true
type	Integer
example	`{ "status": 0 }`

output

description	Output produced by the check `command`.

required	true
type	String
example	<pre>{ "output": "CheckHttp OK: 200, 78572 bytes" }</pre>

source

description	Name of the Sensu entity associated with the event. Use this attribute to tie the event to a proxy entity. If no matching entity exists, Sensu creates a proxy entity with the name provided by the `source` attribute.
required	false
default	The agent entity that receives the event data.
type	String
example	<pre>{ "source": "sensu-docs-site" }</pre>

client

description	Name of the Sensu entity associated with the event. Use this attribute to tie the event to a proxy entity. If no matching entity exists, Sensu creates a proxy entity with the name provided by the `client` attribute. ***NOTE**: The `client` attribute is deprecated in favor of the `source` attribute.*
required	false

default	The agent entity that receives the event data.
type	String
example	<pre>{ "client": "sensu-docs-site" }</pre>

executed

description	Time at which the check was executed. In seconds since the Unix epoch.
required	false
default	The time the event was received by the agent.
type	Integer
example	<pre>{ "executed": 1458934742 }</pre>

duration

description	Amount of time it took to execute the check. In seconds.
required	false
type	Float
example	<pre>{ "duration": 1.903135228 }</pre>

command

description	Command executed to produce the event. Use the `command` attribute to add context to the event data. Sensu does not execute the command included in this attribute.
required	false
type	String
example	<pre>{ "command": "http-check --url https://sensu.io" }</pre>

interval

description	Interval used to produce the event. Use the `interval` attribute to add context to the event data. Sensu does not act on the value provided in this attribute.
required	false
default	`1`
type	Integer
example	<pre>{ "interval": 60 }</pre>

handlers

description	Array of Sensu handler names to use for handling the event. Each handler name in the array must be a string.
required	false

type	Array
example	<pre>{ "handlers": ["slack", "influxdb"] }</pre>

Registration, endpoint management, and service discovery

Sensu agents automatically discover and register infrastructure components and the services running on them. When an agent process stops, the Sensu backend can automatically create and process a deregistration event for the agent entities.

Read Automatically register and deregister entities for more information.

Agent configuration options

Agent configuration is customizable. This section describes each configuration option in more detail, including examples for each configuration method.

You can customize agent configuration with the agent configuration file (Linux and Windows), command line flag arguments (Linux), or environment variables (Linux and Windows).

***NOTE**: The agent loads configuration upon startup, so you must restart the agent for any configuration updates to take effect.*

To view available configuration options for the `sensu-agent start` command, run:

```
sensu-agent start --help
```

The response will list configuration options as command line flags for `sensu-agent start`:

```
```

```
start the sensu agent

Usage:
  sensu-agent start [flags]

Flags:
      --agent-managed-entity                manage this entity via the agent
      --allow-list string                   path to agent execution allow list
configuration file
      --annotations stringToString          entity annotations map (default [])
      --api-host string                     address to bind the Sensu client HTTP
API to (default "127.0.0.1")
      --api-port int                        port the Sensu client HTTP API listens
on (default 3031)
      --assets-burst-limit int              asset fetch burst limit (default 100)
      --assets-rate-limit float             maximum number of assets fetched per
second
      --backend-handshake-timeout int       number of seconds the agent should wait
when negotiating a new WebSocket connection (default 15)
      --backend-heartbeat-interval int      interval at which the agent should send
heartbeats to the backend (default 30)
      --backend-heartbeat-timeout int       number of seconds the agent should wait
for a response to a hearbeat (default 45)
      --backend-url strings                 comma-delimited list of ws/wss URLs of
Sensu backend servers. This flag can also be invoked multiple times (default
[ws://127.0.0.1:8081])
      --cache-dir string                    path to store cached data (default
"/var/cache/sensu/sensu-agent")
      --cert-file string                    certificate for TLS authentication
  -c, --config-file string                  path to sensu-agent config file (default
"/etc/sensu/agent.yml")
      --deregister                          ephemeral agent
      --deregistration-handler string       deregistration handler that should
process the entity deregistration event
      --detect-cloud-provider               enable cloud provider detection
      --disable-api                         disable the Agent HTTP API
      --disable-assets                      disable check assets on this agent
      --disable-sockets                     disable the Agent TCP and UDP event
sockets
      --discover-processes                  indicates whether process discovery
should be enabled
      --events-burst-limit int              /events api burst limit (default 10)
```

```
      --events-rate-limit float             maximum number of events transmitted to
the backend through the /events api
  -h, --help                                help for start
      --insecure-skip-tls-verify            skip TLS verification (not recommended!)
      --keepalive-critical-timeout uint32   number of seconds until agent is
considered dead by backend to create a critical event
      --keepalive-handlers strings          comma-delimited list of keepalive
handlers for this entity. This flag can also be invoked multiple times
      --keepalive-interval int              number of seconds to send between
keepalive events (default 20)
      --keepalive-pipelines strings         comma-delimited list of pipeline
references for keepalive event
      --keepalive-warning-timeout uint32    number of seconds until agent is
considered dead by backend to create a warning event (default 120)
      --key-file string                     key for TLS authentication
      --labels stringToString               entity labels map (default [])
      --log-level string                    logging level [panic, fatal, error,
warn, info, debug] (default "info")
      --max-session-length                  maximum amount of time after which the
agent will reconnect to one of the configured backends (no maximum by default)
      --name string                         agent name (defaults to hostname)
(default "my_hostname")
      --namespace string                    agent namespace (default "default")
      --password string                     agent password (default "P@ssw0rd!")
      --redact strings                      comma-delimited list of fields to redact,
overwrites the default fields. This flag can also be invoked multiple times (default
[password,passwd,pass,api_key,api_token,access_key,secret_key,private_key,secret])
      --require-fips                        indicates whether fips support should be
required in openssl
      --require-openssl                     indicates whether openssl should be
required instead of go's built-in crypto
      --retry-max                           maximum amount of time to wait before
retrying an agent connection to the backend
      --retry-min                           minimum amount of time to wait before
retrying an agent connection to the backend
      --retry-multiplier                    value multiplied with the current retry
delay to produce a longer retry delay (bounded by --retry-max)
      --socket-host string                  address to bind the Sensu client socket
to (default "127.0.0.1")
      --socket-port int                     port the Sensu client socket listens on
(default 3030)
      --statsd-disable                      disables the statsd listener and metrics
```

```
server
      --statsd-event-handlers strings       comma-delimited list of event handlers
for statsd metrics. This flag can also be invoked multiple times
      --statsd-flush-interval int           number of seconds between statsd flush
(default 10)
      --statsd-metrics-host string          address used for the statsd metrics
server (default "127.0.0.1")
      --statsd-metrics-port int             port used for the statsd metrics server
(default 8125)
      --subscriptions strings               comma-delimited list of agent
subscriptions. This flag can also be invoked multiple times
      --trusted-ca-file string              TLS CA certificate bundle in PEM format
      --user string                         agent user (default "agent")
```

***NOTE**: Process discovery is disabled in this version of Sensu. The `discover-processes` configuration option is not available, and new events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.*

General configuration

agent-managed-entity	
description	Indicates whether the agent's entity solely managed by the agent rather than the backend API. Agent-managed entity definitions will include the label `sensu.io/managed_by: sensu-agent`, and you cannot update these agent-managed entities via the Sensu backend REST API.
required	false
type	Boolean
default	false
environment variable	`SENSU_AGENT_MANAGED_ENTITY`
command line example	`sensu-agent start --agent-managed-entity`

agent.yml config file example	`agent-managed-entity: true`

allow-list

description	Path to yaml or json file that contains the allow list of check or hook commands the agent can execute. Read Allow list configuration and the example allow list configuration for information about building a configuration file.
type	String
default	`""`
environment variable	`SENSU_ALLOW_LIST`
command line example	`sensu-agent start --allow-list /etc/sensu/check-allow-list.yaml`
agent.yml config file example	`allow-list: /etc/sensu/check-allow-list.yaml`

annotations

description	Non-identifying metadata to include with event data that you can access with event filters and tokens. You can use annotations to add data that is meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI view filtering. ***NOTE**: For annotations that you define in agent.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the annotation `webhookURL: "https://my-webhook.com"` in agent.yml, it will be listed as `webhookurl:`*

`"https://my-webhook.com"` in entity definitions.

*Key cases are **not** modified for annotations you define with the `--annotations` command line flag or the `SENSU_ANNOTATIONS` environment variable.*

required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_ANNOTATIONS`
command line example	`sensu-agent start --annotations sensu.io/plugins/slack/config/webhook-url=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXX` `sensu-agent start --annotations example-key="example value" --annotations example-key2="example value"`
agent.yml config file example	`annotations:` `  sensu.io/plugins/slack/config/webhook-url: "https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXX"`

assets-burst-limit

description	Maximum amount of burst allowed in a rate interval when fetching dynamic runtime assets.
type	Integer
default	`100`
environment variable	`SENSU_ASSETS_BURST_LIMIT`

command line example	`sensu-agent start --assets-burst-limit 100`
agent.yml config file example	`assets-burst-limit: 100`

assets-rate-limit

description	Maximum number of dynamic runtime assets to fetch per second. The default value `1.39` is equivalent to approximately 5000 user-to-server requests per hour.
type	Float
default	`1.39`
environment variable	`SENSU_ASSETS_RATE_LIMIT`
command line example	`sensu-agent start --assets-rate-limit 1.39`
agent.yml config file example	`assets-rate-limit: 1.39`

backend-handshake-timeout

description	Number of seconds the Sensu agent should wait when negotiating a new WebSocket connection.
type	Integer
default	`15`
environment variable	`SENSU_BACKEND_HANDSHAKE_TIMEOUT`
command line example	

	`sensu-agent start --backend-handshake-timeout 20`
agent.yml config file example	`backend-handshake-timeout: 20`

backend-heartbeat-interval

description	Interval at which the agent should send heartbeats to the Sensu backend. In seconds.
type	Integer
default	`30`
environment variable	`SENSU_BACKEND_HEARTBEAT_INTERVAL`
command line example	`sensu-agent start --backend-heartbeat-interval 45`
agent.yml config file example	`backend-heartbeat-interval: 45`

backend-heartbeat-timeout

description	Number of seconds the agent should wait for a response to a hearbeat from the Sensu backend.
type	Integer
default	`45`
environment variable	`SENSU_BACKEND_HEARTBEAT_TIMEOUT`
command line example	`sensu-agent start --backend-heartbeat-timeout 60`

agent.yml config file example	`backend-heartbeat-timeout: 60`

backend-url

description	ws or wss URL of the Sensu backend server. To specify multiple backends with `sensu-agent start`, use this flag multiple times. ***NOTE**: If you do not specify a port for your backend-url values, the agent will automatically append the default backend port (8081).*
type	List
default	`ws://127.0.0.1:8081` (Debian and RHEL families) `$SENSU_HOSTNAME:8080` (Docker) ***NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.*
environment variable	`SENSU_BACKEND_URL`

SHELL

command line example	`sensu-agent start --backend-url ws://127.0.0.1:8081` `sensu-agent start --backend-url ws://127.0.0.1:8081 --backend-url ws://127.0.0.1:8082`

SHELL

`sensu-agent start --backend-url wss://127.0.0.1:8081`
`sensu-agent start --backend-url wss://127.0.0.1:8081 --backend-url wss://127.0.0.1:8082`

SHELL

agent.yml config file example	``` backend-url: - "ws://127.0.0.1:8081" - "ws://127.0.0.1:8082" ```

SHELL

```
backend-url:
  - "wss://127.0.0.1:8081"
  - "wss://127.0.0.1:8082"
```

cache-dir

description	Path to store cached data.
type	String
default	¬ Linux: `/var/cache/sensu/sensu-agent` ¬ Windows: `C:\ProgramData\sensu\cache\sensu-agent`
environment variable	`SENSU_CACHE_DIR`
command line example	`sensu-agent start --cache-dir /cache/sensu-agent`
agent.yml config file example	`cache-dir: "/cache/sensu-agent"`

config-file

description	Path to Sensu agent configuration file.
type	String

default	¬ Linux: `/etc/sensu/agent.yml` ¬ FreeBSD: `/usr/local/etc/sensu/agent.yml` ¬ Windows: `C:\ProgramData\sensu\config\agent.yml`
environment variable	`SENSU_CONFIG_FILE`
command line example	`sensu-agent start --config-file /sensu/agent.yml` `sensu-agent start -c /sensu/agent.yml`

disable-assets

description	When set to `true`, disables dynamic runtime assets for the agent. If an agent attempts to execute a check that requires a dynamic runtime asset, the agent will respond with a status of `3` and a message that indicates the agent could not execute the check because assets are disabled.
type	Boolean
default	false
environment variable	`SENSU_DISABLE_ASSETS`
command line example	`sensu-agent start --disable-assets`
agent.yml config file example	`disable-assets: true`

discover-processes

description	When set to `true`, the agent populates the `processes` field in `entity.system` and updates every 20 seconds.

***COMMERCIAL FEATURE**: Access the `discover-processes` configuration option in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Process discovery is disabled in this version of Sensu. The discover-processes flag is not available, and new events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.*

type	Boolean
default	false
environment variable	`SENSU_DISCOVER_PROCESSES`
command line example	`sensu-agent start --discover-processes`
agent.yml config file example	`discover-processes: true`

labels

description	Custom attributes to include with event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels. ***NOTE**: For labels that you define in agent.yml, the keys are*

automatically modified to use all lower-case letters. For example, if you define the label `proxyType: "website"` in agent.yml, it will be listed as `proxytype: "website"` in entity definitions.

*Key cases are **not** modified for labels you define with the `--labels` command line flag or the `SENSU_LABELS` environment variable.*

required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_LABELS`
command line example	`sensu-agent start --labels proxy_type=website` `sensu-agent start --labels example_key1="example value" example_key2="example value"`
agent.yml config file example	`labels:` `  proxy_type: website`

log-level

description	Logging level: `panic`, `fatal`, `error`, `warn`, `info`, or `debug`.
type	String
default	`warn`
environment variable	`SENSU_LOG_LEVEL`
command line example	`sensu-agent start --log-level debug`

agent.yml config file example	`log-level: debug`

max-session-length

description	Maximum duration for any one agent connection. In milliseconds (`ms`), seconds (`s`), minutes (`m`), or hours (`h`). Use max-session-length to prevent agent connection distribution from becoming skewed over time. The max-session-length algorithm includes random jitter so that agents will not disconnect and reconnect all at once. Based on the random jitter calculation, at some time before a connection reaches the specified maximum duration, Sensu will force the agent to disconnect and reconnect to an available configured backend.
type	String
default	Defaults to no maximum.
environment variable	`SENSU_MAX_SESSION_LENGTH`
command line example	`sensu-agent start --max-session-length 15m`
agent.yml config file example	`max-session-length: 15m`

name

description	Entity name assigned to the agent entity.
type	String
default	Defaults to hostname (for example, `sensu-centos`).
environment variable	`SENSU_NAME`

command line example	`sensu-agent start --name agent-01`
agent.yml config file example	`name: "agent-01"`

retry-max

description	Maximum amount of time to wait before retrying an agent connection to the backend. In milliseconds (`ms`), seconds (`s`), minutes (`m`), or hours (`h`).
type	String
default	`120s`
environment variable	`SENSU_RETRY_MAX`
command line example	`sensu-agent start --retry-max 120s`
agent.yml config file example	`retry-max: 120s`

retry-min

description	Minimum amount of time to wait before retrying an agent connection to the backend. Multiplied with the retry-multiplier value at each retry. In milliseconds (`ms`), seconds (`s`), minutes (`m`), or hours (`h`).
type	String
default	`1s`
environment variable	`SENSU_RETRY_MIN`

command line example	`sensu-agent start --retry-min 1s`
agent.yml config file example	`retry-min: 1s`

retry-multiplier

description	Value to multiply with the current retry-min delay to produce longer delays at each retry for exponential backoff. ***NOTE**: The maximum retry delay cannot not exceed the retry-max value.*
type	Float
default	`2.0`
environment variable	`SENSU_RETRY_MULTIPLIER`
command line example	`sensu-agent start --retry-multiplier 2.0`
agent.yml config file example	`retry-multiplier: 2.0`

subscriptions

description	Array of agent subscriptions that determine which monitoring checks the agent will execute. The subscriptions array items must be strings.
type	List
environment variable	`SENSU_SUBSCRIPTIONS`

command line example	`sensu-agent start --subscriptions disk-checks,process-checks` `sensu-agent start --subscriptions disk-checks --subscriptions process-checks`
agent.yml config file example	`subscriptions:` `  - disk-checks` `  - process-checks`

API configuration

api-host

description	Bind address for the Sensu agent HTTP API.
type	String
default	`127.0.0.1`
environment variable	`SENSU_API_HOST`
command line example	`sensu-agent start --api-host 127.0.0.1`
agent.yml config file example	`api-host: "127.0.0.1"`

api-port

description	Listening port for the Sensu agent HTTP API.
type	Integer

default	`3031`
environment variable	`SENSU_API_PORT`
command line example	`sensu-agent start --api-port 3031`
agent.yml config file example	`api-port: 3031`

disable-api

description	`true` to disable the agent HTTP API. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_DISABLE_API`
command line example	`sensu-agent start --disable-api`
agent.yml config file example	`disable-api: true`

events-burst-limit

description	Maximum amount of burst allowed in a rate interval for the agent events API.
type	Integer
default	`10`
environment variable	`SENSU_EVENTS_BURST_LIMIT`

command line example	`sensu-agent start --events-burst-limit 20`
agent.yml config file example	`events-burst-limit: 20`

events-rate-limit

description	Maximum number of events per second that can be transmitted to the backend with the agent events API.
type	Float
default	`10.0`
environment variable	`SENSU_EVENTS_RATE_LIMIT`
command line example	`sensu-agent start --events-rate-limit 20.0`
agent.yml config file example	`events-rate-limit: 20.0`

Ephemeral agent configuration

deregister

description	`true` if a deregistration event should be created upon Sensu agent process stop. Otherwise, `false`. ***NOTE**: To receive alerts for failing keepalives, set to `false`.*

type	Boolean
default	`false`
environment variable	`SENSU_DEREGISTER`
command line example	`sensu-agent start --deregister`
agent.yml config file example	`deregister: true`

deregistration-handler

description	Name of the event handler to use when processing the agent's deregistration events. This configuration option overrides any handlers applied by the `deregistration-handler` backend configuration option.
type	String
environment variable	`SENSU_DEREGISTRATION_HANDLER`
command line example	`sensu-agent start --deregistration-handler deregister`
agent.yml config file example	`deregistration-handler: deregister`

detect-cloud-provider

description	`true` to enable cloud provider detection mechanisms. Otherwise, `false`. When this option is enabled, the agent will attempt to read files, resolve hostnames, and make HTTP requests to determine what cloud environment it is running in.

type	Boolean
default	`false`
environment variable	`SENSU_DETECT_CLOUD_PROVIDER`
command line example	`sensu-agent start --detect-cloud-provider false`
agent.yml config file example	`detect-cloud-provider: false`

Keepalive configuration

keepalive-critical-timeout

description	Number of seconds after a missing keepalive event until the agent is considered unresponsive by the Sensu backend to create a critical event. Set to disabled (`0`) by default. If the value is not `0` , it must be greater than or equal to `5` . ***NOTE**: The agent maps the `keepalive-critical-timeout` value to the `event.check.ttl` attribute when keepalive events are generated for the Sensu backend to process. The `event.check.ttl` attribute is useful for creating time-based event filters to reduce alert fatigue for agent keepalive events.*
type	Integer
default	`0`
environment variable	`SENSU_KEEPALIVE_CRITICAL_TIMEOUT`
command line example	`sensu-agent start --keepalive-critical-timeout 300`

agent.yml config file example	`keepalive-critical-timeout: 300`

keepalive-handlers

description	Keepalive event handlers to use for the entity, specified in a comma-delimited list. You can specify any configured handler and invoke the `keepalive-handlers` configuration option multiple times. If keepalive handlers are not specified, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI.
type	List
default	`keepalive`
environment variable	`SENSU_KEEPALIVE_HANDLERS`
command line example	`sensu-agent start --keepalive-handlers slack,email`
agent.yml config file example	`keepalive-handlers:` `- slack` `- email`

keepalive-interval

description	Number of seconds between keepalive events.
type	Integer
default	`20`
environment variable	`SENSU_KEEPALIVE_INTERNAL`
command line	`sensu-agent start --keepalive-interval 30`

example	

agent.yml config file example	`keepalive-interval: 30`

keepalive-pipelines

description	Pipelines to use for processing keepalive events, specified in a comma-delimited list. If keepalive pipelines are not specified, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI. To specify pipelines for the `keepalive-pipelines` configuration option, use the fully qualified name for pipeline resources (`core/v2.Pipeline`) plus the pipeline name.
type	List
default	`keepalive`
environment variable	`SENSU_KEEPALIVE_PIPELINES`
command line example	`sensu-agent start --keepalive-pipelines core/v2.Pipeline.slack,core/v2.Pipeline.store-keepalives`
agent.yml config file example	`keepalive-pipelines:` `- core/v2.Pipeline.slack` `- core/v2.Pipeline.store-keepalives`

keepalive-warning-timeout

description	Number of seconds after a missing keepalive event until the agent is considered unresponsive by the Sensu backend to create a warning event. Value must be lower than the

`keepalive-critical-timeout` value. Minimum value is `5`.

> ***NOTE**: The agent maps the `keepalive-warning-timeout` value to the `event.check.timeout` attribute when keepalive events are generated for the Sensu backend to process. The `event.check.timeout` attribute is useful for creating time-based event filters to reduce alert fatigue for agent keepalive events.*

type	Integer
default	`120`
environment variable	`SENSU_KEEPALIVE_WARNING_TIMEOUT`
command line example	`sensu-agent start --keepalive-warning-timeout 300`
agent.yml config file example	`keepalive-warning-timeout: 300`

Security configuration

cert-file

description	Path to the agent certificate file used in mTLS authentication. Sensu supports certificate bundles (or chains) as long as the agent (or leaf) certificate is the *first* certificate in the bundle.
type	String
default	`""`
environment variable	`SENSU_CERT_FILE`
command line example	`sensu-agent start --cert-file /path/to/tls/agent.pem`

agent.yml config file example	`cert-file: "/path/to/tls/agent.pem"`

insecure-skip-tls-verify

description	Skip SSL verification. ***WARNING**: This configuration option is intended for use in development systems only. Do not use this configuration option in production.*
type	Boolean
default	`false`
environment variable	`SENSU_INSECURE_SKIP_TLS_VERIFY`
command line example	`sensu-agent start --insecure-skip-tls-verify`
agent.yml config file example	`insecure-skip-tls-verify: true`

key-file

description	Path to the agent key file used in mTLS authentication.
type	String
default	`""`
environment variable	`SENSU_KEY_FILE`

command line example	`sensu-agent start --key-file /path/to/tls/agent-key.pem`
agent.yml config file example	`key-file: "/path/to/tls/agent-key.pem"`

namespace

description	Agent namespace. ***NOTE**: Agents are represented in the backend as a class of entity. Entities can only belong to a single namespace.*
type	String
default	`default`
environment variable	`SENSU_NAMESPACE`
command line example	`sensu-agent start --namespace ops`
agent.yml config file example	`namespace: ops`

password

description	Sensu RBAC password used by the agent.
type	String
default	`P@ssw0rd!`
environment variable	`SENSU_PASSWORD`

command line example	`sensu-agent start --password secure-password`
agent.yml config file example	`password: secure-password`

redact

description	List of fields to redact when displaying the entity. ***NOTE**: Redacted secrets are sent via the WebSocket connection and stored in etcd. They are not logged or displayed via the Sensu API.*
type	List
default	By default, Sensu redacts the following fields: `password`, `passwd`, `pass`, `api_key`, `api_token`, `access_key`, `secret_key`, `private_key`, `secret`.
environment variable	`SENSU_REDACT`
command line example	`sensu-agent start --redact secret,ec2_access_key`
agent.yml config file example	`redact:` `  - secret` `  - ec2_access_key`

require-fips

description	Require Federal Information Processing Standard (FIPS) support in

OpenSSL. Logs an error at Sensu agent startup if `true` but OpenSSL is not running in FIPS mode.

> ***NOTE**: The `require-fips` configuration option is only available within the Linux amd64 OpenSSL-linked binary. Contact Sensu to request the builds for OpenSSL with FIPS support.*

type	Boolean
default	false
environment variable	`SENSU_REQUIRE_FIPS`
command line example	`sensu-agent start --require-fips`
agent.yml config file example	`require-fips: true`

require-openssl

description	Use OpenSSL instead of Go's standard cryptography library. Logs an error at Sensu agent startup if `true` but Go's standard cryptography library is loaded.

> ***NOTE**: The `require-openssl` configuration option is only available within the Linux amd64 OpenSSL-linked binary. Contact Sensu to request the builds for OpenSSL with FIPS support.*

type	Boolean
default	false
environment variable	`SENSU_REQUIRE_OPENSSL`
command line	

example	`sensu-agent start --require-openssl`
agent.yml config file example	`require-openssl: true`

trusted-ca-file

description	SSL/TLS certificate authority.
type	String
default	`""`
environment variable	`SENSU_TRUSTED_CA_FILE`
command line example	`sensu-agent start --trusted-ca-file /path/to/tls/ca.pem`
agent.yml config file example	`trusted-ca-file: "/path/to/tls/ca.pem"`

user

description	Sensu RBAC username used by the agent. Agents require get, list, create, update, and delete permissions for events across all namespaces.
type	String
default	`agent`
environment variable	`SENSU_USER`
command line example	`sensu-agent start --user agent-01`

agent.yml config file example	`user: "agent-01"`

Socket configuration

disable-sockets

description	`true` to disable the agent TCP and UDP event sockets. Othewise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_DISABLE_SOCKETS`
command line example	`sensu-agent start --disable-sockets`
agent.yml config file example	`disable-sockets: true`

socket-host

description	Address to bind the Sensu agent socket to.
type	String
default	`127.0.0.1`
environment variable	`SENSU_SOCKET_HOST`
command line example	`sensu-agent start --socket-host 127.0.0.1`

agent.yml config file example	`socket-host: "127.0.0.1"`

socket-port	
description	Port the Sensu agent socket listens on.
type	Integer
default	`3030`
environment variable	`SENSU_SOCKET_PORT`
command line example	`sensu-agent start --socket-port 3030`
agent.yml config file example	`socket-port: 3030`

StatsD configuration

statsd-disable	
description	`true` to disable the StatsD listener and metrics server. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_STATSD_DISABLE`
command line example	`sensu-agent start --statsd-disable`

agent.yml config file example	`statsd-disable: true`

statsd-event-handlers

description	List of event handlers for StatsD metrics.
type	List
environment variable	`SENSU_STATSD_EVENT_HANDLERS`
command line example	`sensu-agent start --statsd-event-handlers influxdb,opentsdb` `sensu-agent start --statsd-event-handlers influxdb --statsd-event-handlers opentsdb`
agent.yml config file example	`statsd-event-handlers:` `  - influxdb` `  - opentsdb`

statsd-flush-interval

description	Number of seconds between StatsD flushes.
type	Integer
default	`10`
environment variable	`SENSU_STATSD_FLUSH_INTERVAL`
command line example	`sensu-agent start --statsd-flush-interval 30`
agent.yml config file	

example	`statsd-flush-interval: 30`

statsd-metrics-host

description	Address used for the StatsD metrics server.
type	String
default	`127.0.0.1`
environment variable	`SENSU_STATSD_METRICS_HOST`
command line example	`sensu-agent start --statsd-metrics-host 127.0.0.1`
agent.yml config file example	`statsd-metrics-host: "127.0.0.1"`

statsd-metrics-port

description	Port used for the StatsD metrics server.
type	Integer
default	`8125`
environment variable	`SENSU_STATSD_METRICS_PORT`
command line example	`sensu-agent start --statsd-metrics-port 8125`
agent.yml config file example	`statsd-metrics-port: 8125`

Allow list configuration

The allow list includes check and hook commands the agent can execute. Use the `allow-list` configuration option to specify the path to the yaml or json file that contains your allow list.

Use these commands to build your allow list configuration file.

args	
description	Arguments for the `exec` command.
required	true
type	Array
example	**YAML** `args:` `- foo` **JSON** `{` `  "args": ["foo"]` `}`

enable_env	
description	`true` to enable environment variables. Otherwise, `false`.
required	false
type	Boolean
example	**YAML** `enable_env: true` **JSON**

```
{
  "enable_env": true
}
```

exec

description	Command to allow the Sensu agent to run as a check or a hook.
required	true
type	String
example	**YML** `exec: "/usr/local/bin/check_memory.sh"` **JSON** `{ "exec": "/usr/local/bin/check_memory.sh" }`

sha512

description	Checksum of the check or hook executable.
required	false
type	String
example	**YML** `sha512: 4f926bf4328...` **JSON**

```
{
  "sha512": "4f926bf4328..."
}
```

Example allow list configuration

YML

```
- exec: /usr/local/bin/check_memory.sh
  args:
  - ""
  sha512:
736ac120323772543fd3a08ee54afdd54d214e58c280707b63ce652424313ef9084ca5b247d226aa09be
8f831034ff4991bfb95553291c8b3dc32cad034b4706
  enable_env: true
  foo: bar
- exec: /usr/local/bin/show_process_table.sh
  args:
  - ""
  sha512:
28d61f303136b16d20742268a896bde194cc99342e02cdffc1c2186f81c5adc53f8550635156bebeed7d
87a0c19a7d4b7a690f1a337cc4737e240b62b827f78a
- exec: echo-asset.sh
  args:
  - "foo"
  sha512:
cce3d16e5881ba829f271df778f9014f7c3659917f7acfd7a60a91bfcabb472eea72f9781194d310388b
a046c21790364ad0308a5a897cde50022195ba90924b
```

JSON

```
[
  {
    "exec": "/usr/local/bin/check_memory.sh",
    "args": [
      ""
    ],
    "sha512":
"736ac120323772543fd3a08ee54afdd54d214e58c280707b63ce652424313ef9084ca5b247d226aa09b
```

```
e8f831034ff4991bfb95553291c8b3dc32cad034b4706",
    "enable_env": true,
    "foo": "bar"
  },
  {
    "exec": "/usr/local/bin/show_process_table.sh",
    "args": [
      ""
    ],
    "sha512":
"28d61f303136b16d20742268a896bde194cc99342e02cdffc1c2186f81c5adc53f8550635156bebeed7
d87a0c19a7d4b7a690f1a337cc4737e240b62b827f78a"
  },
  {
    "exec": "echo-asset.sh",
    "args": [
      "foo"
    ],
    "sha512":
"cce3d16e5881ba829f271df778f9014f7c3659917f7acfd7a60a91bfcabb472eea72f9781194d310388
ba046c21790364ad0308a5a897cde50022195ba90924b"
  }
]
```

Agent configuration methods

Agent configuration file

For Linux and Windows agents, you can customize the agent configuration in a `.yml` configuration file.

The default agent configuration file path for Linux is `/etc/sensu/agent.yml`. The default agent configuration file path for Windows is `C:\ProgramData\sensu\config\agent.yml.example`.

To use the `agent.yml` file to configure the agent, list the desired configuration attributes and values. Review the example Sensu agent configuration file for a complete example.

***NOTE**: The agent loads configuration upon startup. If you make changes in the `agent.yml`*

configuration file after startup, you must restart the agent for the changes to take effect.

Configuration via command line flags or environment variables overrides any configuration specified in the agent configuration file.Read Create overrides to learn more.

Command line flags

For Linux agents, you can customize the agent configuration with `sensu-agent start` command line flags.

To use command line flags, specify the desired configuration options and values along with the `sensu-agent start` command.For example:

```
sensu-agent start --name webserver_05 --keepalive-warning-timeout 60 --keepalive-critical-timeout 120
```

Configuration via command line flags overrides attributes specified in a configuration file or with environment variables.Read Create overrides to learn more.

Environment variables

Instead of using the agent configuration file or command line flags, you can use environment variables to configure your Sensu agent.Each agent configuration option has an associated environment variable You can also create your own environment variables, as long as you name them correctly and save them in the correct place.Here's how.

1. Create the files from which the `sensu-agent` service configured by our supported packages will read environment variables:

   SHELL

   ```
   sudo touch /etc/default/sensu-agent
   ```

   SHELL

   ```
   sudo touch /etc/sysconfig/sensu-agent
   ```

SHELL

```
# By default, the agent loads configuration from
%ALLUSERSPROFILE%\sensu\config\agent.yml.
# If you did not change the location for the configuration file during
installation,
# the sensu-agent configuration file path is:

C:\ProgramData\sensu\config\agent.yml
```

2. Make sure the environment variable is named correctly. All environment variables that control Sensu agent configuration begin with `SENSU_`.

   To rename a configuration option you wish to specify as an environment variable, prepend `SENSU_`, convert dashes to underscores, and capitalize all letters. For example, the environment variable for the flag `api-host` is `SENSU_API_HOST`.

   For a custom environment variable, you do not have to prepend `SENSU`. For example, `TEST_VAR_1` is a valid custom environment variable name.

3. Add the environment variable to the environment file.

   In this example, the `api-host` flag is configured as an environment variable and set to `"0.0.0.0"`:

   SHELL

   ```
   echo 'SENSU_API_HOST="0.0.0.0"' | sudo tee -a /etc/default/sensu-agent
   ```

   SHELL

   ```
   echo 'SENSU_API_HOST="0.0.0.0"' | sudo tee -a /etc/sysconfig/sensu-agent
   ```

   SHELL

   ```
   # Save the following environment variable in the configuration file
   # at C:\ProgramData\sensu\config\agent.yml:

   SENSU_API_HOST="0.0.0.0"
   ```

4. Restart the sensu-agent service so these settings can take effect:

SHELL

```
sudo systemctl restart sensu-agent
```

SHELL

```
sudo systemctl restart sensu-agent
```

SHELL

```
sc.exe start SensuAgent
```

***NOTE**: Sensu includes an environment variable for each agent configuration option. They are listed in the configuration description tables.*

Format for label and annotation environment variables

To use labels and annotations as environment variables in your check and plugin configurations, you must use a specific format when you create the environment variables.

For example, to create the labels `"region": "us-east-1"` and `"type": "website"` as an environment variable:

SHELL

```
echo 'SENSU_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo tee -a /etc/default/sensu-agent
```

SHELL

```
echo 'SENSU_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo tee -a /etc/sysconfig/sensu-agent
```

To create the annotations `"maintainer": "Team A"` and `"webhook-url": "https://hooks.slack.com/services/T0000/B00000/XXXXX"` as an environment variable:

SHELL

```
echo 'SENSU_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/default/sensu-agent
```

SHELL

```
echo 'SENSU_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/sysconfig/sensu-agent
```

Use environment variables with the Sensu agent

Any environment variables you create in `/etc/default/sensu-agent` (Debian family) or `/etc/sysconfig/sensu-agent` (RHEL family) will be available to check and hook commands executed by the Sensu agent. This includes your checks and plugins.

For example, if you create a custom environment variable `TEST_VARIABLE` in your sensu-agent file, it will be available to use in your check and hook configurations as `$TEST_VARIABLE`.

The following check example demonstrates how to use a `TEST_GITHUB_TOKEN` environment variable (set to the token value in the sensu-agent file) in the check command to run a script that pings the GitHub API:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: ping-github-api
spec:
  command: ping-github-api.sh $TEST_GITHUB_TOKEN
  handlers:
  - slack
```

```
  interval: 10
  publish: true
  subscriptions:
  - system
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "ping-github-api"
  },
  "spec": {
    "command": "ping-github-api.sh $TEST_GITHUB_TOKEN",
    "handlers": [
      "slack"
    ],
    "interval": 10,
    "publish": true,
    "subscriptions": [
      "system"
    ]
  }
}
```

Use environment variables to specify an HTTP proxy for agent use

If an HTTP proxy is required to access the internet in your compute environment, you may need to configure the Sensu agent to successfully download dynamic runtime assets or execute commands that depend on internet access.

For Sensu agents that require a proxy server, define `HTTP_PROXY` and `HTTPS_PROXY` environment variables in your sensu-agent file.

```
HTTP_PROXY="http://YOUR_PROXY_SERVER:PORT"
HTTPS_PROXY="http://YOUR_PROXY_SERVER:PORT"
```

You can use the same proxy server URL for `HTTP_PROXY` and `HTTPS_PROXY`. The proxy server URL you specify for `HTTPS_PROXY` does not need to use `https://`.

After you add the `HTTP_PROXY` and `HTTPS_PROXY` environment variables and restart sensu-agent, they will be available to check and hook commands executed by the Sensu agent. You can then use `HTTP_PROXY` and `HTTPS_PROXY` to add dynamic runtime assets, run checks, and complete other tasks that typically require an internet connection for your unconnected entities.

***NOTE**: If you define the `HTTP_PROXY` and `HTTPS_PROXY` environment variables, the agent WebSocket connection will also use the proxy URL you specify.*

Create configuration overrides

Sensu has default settings and limits for certain configuration attributes, like the default log level. Depending on your environment and preferences, you may want to create overrides for these Sensu-specific defaults and limits.

You can create configuration overrides in several ways:

- Command line configuration flag arguments for `sensu-agent start`.
- Environment variables in `/etc/default/sensu-agent` (Debian family) or `/etc/sysconfig/sensu-agent` (RHEL family).
- Configuration settings in the agent.yml config file.

***NOTE**: We do not recommend editing the systemd unit file to create overrides. Future package upgrades can overwrite changes in the systemd unit file.*

Sensu applies the following precedence to override settings:

1. Arguments passed to the Sensu agent via command line configuration flags.
2. Environment variables in `/etc/default/sensu-agent` (Debian family) or `/etc/sysconfig/sensu-agent` (RHEL family).
3. Configuration in the agent.yml config file.

For example, if you create overrides using all three methods, the command line configuration flag values will take precedence over the values you specify in `/etc/default/sensu-agent` or `/etc/sysconfig/sensu-agent` or the agent.yml config file.

Example override: Log level

The default log level for the Sensu agent is `warn`.

To override the default and automatically apply a different log level for the agent, add the `--log-level` command line configuration flag when you start the Sensu agent. For example, to specify `debug` as the log level:

```
sensu-agent start --log-level debug
```

To configure an environment variable for the desired agent log level:

SHELL

```
echo 'SENSU_LOG_LEVEL=debug' | sudo tee -a /etc/default/sensu-agent
```

SHELL

```
echo 'SENSU_LOG_LEVEL=debug' | sudo tee -a /etc/sysconfig/sensu-agent
```

To configure the desired log level in the config file, add this line to agent.yml:

```
log-level: debug
```

Service management

> ***NOTE**: Service management commands may require administrative privileges.*

Start the service

Use the `sensu-agent` tool to start the agent and apply configuration flags.

Linux

Start the agent with configuration flags:

```
sensu-agent start --subscriptions disk-checks --log-level debug
```

View available configuration flags and defaults:

```
sensu-agent start --help
```

Start the agent using a service manager:

```
sudo systemctl start sensu-agent
```

If you do not provide any configuration flags, the agent loads configuration from the location specified by the `config-file` attribute (default is `/etc/sensu/agent.yml`).

Windows

Run the following command as the admin user to install and start the agent:

```
sensu-agent service install
```

By default, the agent loads configuration from `%ALLUSERSPROFILE%\sensu\config\agent.yml` (for example, `C:\ProgramData\sensu\config\agent.yml`) and stores service logs to `%ALLUSERSPROFILE%\sensu\log\sensu-agent.log` (for example, `C:\ProgramData\sensu\log\sensu-agent.log`).

Configure the configuration file and log file locations using the `config-file` and `log-file` flags:

```
sensu-agent service install --config-file 'C:\\ProgramData\\sensu\\config\\agent.yml' --log-file 'C:\\ProgramData\\sensu\\log\\sensu-agent.log'
```

Stop the service

Stop the agent service using a service manager:

SHELL

```
sudo systemctl stop sensu-agent
```

SHELL

```
sc.exe stop SensuAgent
```

Restart the service

You must restart the agent to implement any configuration updates.

Linux

Restart the agent with a service manager:

```
sudo systemctl restart sensu-agent
```

Windows

Restart the agent with a service manager:

```
sc.exe start SensuAgent
```

As of Sensu Go 6.8.2, the Sensu Agent service on Windows platforms will automatically restart after failures. You'll still need to use a service manager restart Windows agents to implement configuration updates.

Enable on boot

Enable the agent to start on system boot:

SHELL

```
sudo systemctl enable sensu-agent
```

SHELL

```
The service is configured to start automatically on boot by default.
```

***NOTE**: On older distributions of Linux, use `sudo chkconfig sensu-agent on` to enable the agent.*

Disable on boot

Disable the agent from starting on system boot:

SHELL

```
sudo systemctl disable sensu-agent
```

SHELL

```
The service is configured to start automatically on boot by default.
```

***NOTE**: On older distributions of Linux, use `sudo chkconfig sensu-agent off` to disable the agent.*

Get service status

View the status of the agent service using a service manager:

SHELL

```
sudo systemctl status sensu-agent
```

SHELL

```
sc.exe query SensuAgent
```

Get service version

Get the version of the current `sensu-agent` tool:

```
sensu-agent version
```

Get the version of the running `sensu-agent` service:

```
curl http://127.0.0.1:3031/version
```

Uninstall the service

Uninstall the sensu-agent service:

SHELL

```
sudo systemctl stop sensu-agent
```

SHELL

```
sensu-agent service uninstall
```

Get help

The `sensu-agent` tool provides general and command-specific help flags.

View sensu-agent commands:

```
sensu-agent help
```

List options for a specific command (in this case, `sensu-agent start`):

```
sensu-agent start --help
```

Backend reference

Example Sensu backend configuration file (download)

The Sensu backend is a service that manages check requests and observability data. Every Sensu backend includes an integrated structure for scheduling checks using subscriptions, an event processing pipeline that applies event filters, mutators, handlers, and pipelines, an embedded etcd datastore for storing configuration and state, and the Sensu API, Sensu web UI, and sensuctl command line tool.

The Sensu backend is available for Debian- and RHEL-family distributions of Linux. For these operating systems, the Sensu backend uses the Bourne shell (sh) for the execution environment.

Read the installation guide to install the backend.

Initialization

For a **new** installation, the backend database must be initialized by providing a username and password for the user to be granted administrative privileges. Although initialization is required for every new installation, the implementation differs depending on your method of installation:

- If you are using Docker, you can use environment variables to override the default admin username (`admin`) and password (`P@ssw0rd!`) during step 2 of the backend installation process.
- If you are using a Debian- or RHEL-family distribution, you must specify admin credentials during step 3 of the backend installation process. Sensu does not apply default admin credentials for Debian- or RHEL-family installations.

The initialization step bootstraps the first admin user account for your Sensu installation. This first account will be granted the cluster admin role.

***WARNING**: If you plan to run a Sensu cluster, make sure that each of your backend nodes is configured, running, and a member of the cluster before you initialize.*

Docker initialization

For Docker installations, set administrator credentials with environment variables when you configure and start the backend as shown below.Replace `<username>` and `<password>` with the username and password you want to use:

DOCKER

```
docker run -v /var/lib/sensu:/var/lib/sensu \
-d --name sensu-backend \
-p 3000:3000 -p 8080:8080 -p 8081:8081 \
-e SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username> \
-e SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password> \
sensu/sensu:latest \
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

DOCKER

```
---
version: "3"
services:
  sensu-backend:
    ports:
    - 3000:3000
    - 8080:8080
    - 8081:8081
    volumes:
    - "sensu-backend-data:/var/lib/sensu/sensu-backend/etcd"
    command: "sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-
level debug"
    environment:
    - SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
    - SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
    image: sensu/sensu:latest

volumes:
  sensu-backend-data:
    driver: local
```

If you did not use environment variables to override the default admin credentials in step 2 of the

backend installation process, we recommend changing your default admin password as soon as you have installed sensuctl.

Debian or RHEL family initialization

For Debian- or RHEL-family distributions, set administrator credentials with environment variables at initialization as shown below.

To initialize with your username and password, replace `<username>` and `<password>` with the username and password you want to use:

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
sensu-backend init
```

***NOTE**: Make sure the Sensu backend is running before you run `sensu-backend init`.*

Add API key for initialization

Add an API key when you initialize the backend to make automated cluster setup and deployment more straightforward. An API key is a persistent UUID that maps to a stored Sensu username.

If you supply an API key via sensu-backend init, you do not need to configure sensuctl. Instead, you can execute sensuctl commands to manage resources immediately after initializing a cluster by providing the `--api-key` and `--api-url` flags with their correct values in your sensuctl commands.

To initialize with an API key in addition to username and password, set your administrator credentials as follows. Replace `<api_key>` with the API key you want to use:

DOCKER

```
docker run -v /var/lib/sensu:/var/lib/sensu \
-d --name sensu-backend \
-p 3000:3000 -p 8080:8080 -p 8081:8081 \
-e SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username> \
-e SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password> \
-e SENSU_BACKEND_CLUSTER_ADMIN_API_KEY=<api_key> \
sensu/sensu:latest \
```

```
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

DOCKER

```
---
version: "3"
services:
  sensu-backend:
    ports:
    - 3000:3000
    - 8080:8080
    - 8081:8081
    volumes:
    - "sensu-backend-data:/var/lib/sensu/sensu-backend/etcd"
    command: "sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-
level debug"
    environment:
    - SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
    - SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
    - SENSU_BACKEND_CLUSTER_ADMIN_API_KEY=<api_key>
    image: sensu/sensu:latest

volumes:
  sensu-backend-data:
    driver: local
```

SHELL

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
export SENSU_BACKEND_CLUSTER_ADMIN_API_KEY=<api_key>
sensu-backend init
```

Initialize in interactive mode

You can also run the `sensu-backend init` command in interactive mode:

```
sensu-backend init --interactive
```

You will receive prompts for username, password, and API key in interactive mode. Provide your username and password to complete initialization. The API key is optional — press `return` to skip it.

```
Cluster Admin Username: <username>
Cluster Admin Password: <password>
Retype Cluster Admin Password: <password>
Cluster Admin API Key: <api_key>
```

***NOTE**: If you are already using Sensu, you do not need to initialize. Your installation has already seeded the admin username and password you have set up. Running `sensu-backend init` on a previously initialized cluster has no effect — it will not change the admin credentials.*

Initialization flags

To view available initialization flags:

```
sensu-backend init --help
```

The response will list command information and configuration flags for `sensu-backend init`:

```
Usage:
  sensu-backend init [flags]

General Flags:
      --cluster-admin-api-key string    cluster admin API key
      --cluster-admin-password string   cluster admin password
      --cluster-admin-username string   cluster admin username
  -c, --config-file string              path to sensu-backend config file (default
"/etc/sensu/backend.yml")
  -h, --help                            help for init
      --ignore-already-initialized      exit 0 if the cluster has already been
initialized
      --interactive                     interactive mode
```

```
      --timeout string                  duration to wait before a connection attempt
to etcd is considered failed (must be >= 1s) (default "5s")
      --wait                            continuously retry to establish a connection
to etcd until it is successful

Store Flags:
      --etcd-advertise-client-urls strings   list of this member's client URLs to
advertise to clients (default [http://localhost:2379])
      --etcd-cert-file string                path to the client server TLS cert file
      --etcd-cipher-suites strings           list of ciphers to use for etcd TLS
configuration
      --etcd-client-cert-auth                enable client cert authentication
      --etcd-client-urls string              client URLs to use when operating as an
etcd client
      --etcd-key-file string                 path to the client server TLS key file
      --etcd-max-request-bytes uint          maximum etcd request size in bytes (use
with caution) (default 1572864)
      --etcd-trusted-ca-file string          path to the client server TLS trusted CA
cert file
```

For more information about the initialization store flags, read Datastore and cluster configuration and Advanced configuration options.

ignore-already-initialized

description	If you run sensu-backend init on a cluster that has already been initialized, the command returns a non-zero exit status. Add the `ignore-already-initialized` flag to suppress the “already initialized” response and return an exit code 0 if the cluster has already been initialized.
example	`sensu-backend init --ignore-already-initialized`

timeout

description	Specify how long the backend should continue trying to establish a connection to etcd before timing out.

To specify the timeout duration, use an integer paired with a unit of time: `s` for seconds, `m` for minutes, or `h` for hours.

> ***NOTE**: Sensu interprets timeout values less than 1 second and integer-only values as seconds. For example, Sensu will convert both `20ms` and `20` to `20s` (20 seconds).*

type	String
example	`sensu-backend init --timeout 30s`

wait	
description	Instruct the backend to continue trying to establish a connection to etcd until it is successful.
example	`sensu-backend init --wait`

Backend transport

The Sensu backend listens for agent communications via WebSocket transport.By default, this transport operates on port 8081.The agent subscriptions are used to determine which check execution requests the backend publishes via the transport.Sensu agents locally execute checks as requested by the backend and publish check results back to the transport to be processed.

Sensu agents authenticate to the Sensu backend via transport by either built-in username and password authentication or mutual transport layer security (mTLS) authentication.

To secure the WebSocket transport, first generate the certificates you will need to set up transport layer security (TLS).Then, secure Sensu by configuring either TLS or mTLS to make Sensu production-ready.

Read the Sensu architecture overview for a diagram that includes the WebSocket transport.

Certificate bundles or chains

The Sensu backend supports all types of certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle. This is because the Go standard library assumes that the first certificate listed in the PEM file is the server certificate — the certificate that the program will use to show its own identity.

If you send the server certificate alone instead of sending the whole bundle with the server certificate first, you will receive a `certificate not signed by trusted authority` error. You must present the whole chain to the remote so it can determine whether it trusts the server certificate through the chain.

Certificate revocation check

The Sensu backend checks certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP) endpoints for mutual transport layer security (mTLS), etcd client, and etcd peer connections whose remote sides present X.509 certificates that provide CRL and OCSP revocation information.

Startup and backend entities

When a backend starts up, Sensu automatically checks for a `sensu-system` namespace (and creates the namespace if it doesn't exist). Then, Sensu checks the `sensu-system` namespace for an existing entity named after the backend's local hostname.

- If there is no corresponding entity, Sensu creates a new entity with `entity_class: backend` and populates the entity's system information.
- If there is a corresponding entity, Sensu does nothing further to the existing entity.

Once the backend entity is created, the backend uses its own entity to report cluster state errors. Read backend entities in the entities reference for more information and an example backend entity definition.

Synchronize time between agents and the backend

System clocks between agents and the backend should be synchronized to a central NTP server. If system time is out of sync, it may cause issues with keepalive, metric, and check alerts.

Backend clusters

You can run the backend as a standalone service, but running a cluster of backends makes Sensu more highly available, reliable, and durable. Sensu backend clusters build on the etcd clustering system Clustering lets you synchronize data between backends and get the benefits of a highly available configuration.

To configure a cluster, read Run a Sensu cluster and review the datastore configuration options.

Create event pipelines

Sensu backend event pipelines process observation data and execute event filters, mutators, handlers, and pipelines. These resources are powerful tools to automate your monitoring workflows.

Read the event filter, mutator, handler, and pipeline references to learn more about these Sensu resources. Read guides like Reduce alert fatigue with event filters and Send Slack alerts with handlers for usage examples.

Schedule checks

The backend is responsible for storing check definitions and scheduling check requests. Check scheduling is subscription-based: the backend sends check requests to subscriptions, where they're picked up by subscribing agents.

For information about creating and managing checks, read the checks reference and the guides Monitor server resources with checks and Collect metrics with checks.

Backend configuration options

Backend configuration is customizable. This section describes each configuration option in more detail, including examples for each configuration method.

You can customize backend configuration with the backend configuration file, command line flag arguments, or environment variables.

***NOTE**: The backend loads configuration upon startup, so you must restart the backend for any configuration updates to take effect.*

To view configuration information for the `sensu-backend start` command, run:

```
sensu-backend start --help
```

The response will list configuration options as command line flags for `sensu-backend start`:

```
start the sensu backend

Usage:
  sensu-backend start [flags]

General Flags:
      --agent-auth-cert-file string              TLS certificate in PEM format for
agent certificate authentication
      --agent-auth-crl-urls strings               URLs of CRLs for agent certificate
authentication
      --agent-auth-key-file string               TLS certificate key in PEM format
for agent certificate authentication
      --agent-auth-trusted-ca-file string        TLS CA certificate bundle in PEM
format for agent certificate authentication
      --agent-burst-limit int                     agent connections maximum burst
size
      --agent-host string                         agent listener host (default "
[::]")
      --agent-serve-wait-time duration            wait time before accepting agent
connections on startup
      --agent-port int                            agent listener port (default 8081)
      --agent-rate-limit int                      agent connections maximum rate
limit
      --agent-write-timeout int                   timeout in seconds for agent
writes (default 15)
      --annotations stringToString                entity annotations map (default
[])
      --api-listen-address string                 address to listen on for api
traffic (default "[::]:8080")
      --api-serve-wait-time duration              wait time before serving API
```

```
requests on startup
      --api-request-limit int                    maximum API request body size, in
bytes (default 512000)
      --api-url string                           url of the api to connect to
(default "http://localhost:8080")
      --api-write-timeout                        maximum duration before timing out
writes of responses
      --assets-burst-limit int                   asset fetch burst limit (default
100)
      --assets-rate-limit float                  maximum number of assets fetched
per second
      --cache-dir string                         path to store cached data (default
"/var/cache/sensu/sensu-backend")
      --cert-file string                         TLS certificate in PEM format
  -c, --config-file string                       path to sensu-backend config file
(default "/etc/sensu/backend.yml")
      --dashboard-cert-file string               dashboard TLS certificate in PEM
format
      --dashboard-host string                    dashboard listener host (default "
[::]")
      --dashboard-key-file string                dashboard TLS certificate key in PEM
format
      --dashboard-port int                       dashboard listener port (default
3000)
      --dashboard-write-timeout                  maximum duration before timing out
writes of responses
      --debug                                    enable debugging and profiling
features
      --deregistration-handler string            default deregistration handler
      --disable-platform-metrics                 disable platform metrics logging
      --event-log-buffer-size int                buffer size of the event logger
(default 100000)
      --event-log-buffer-wait string             full buffer wait time (default
"10ms")
      --event-log-file string                    path to the event log file
      --event-log-parallel-encoders              used to indicate parallel encoders
should be used for event logging
      --eventd-buffer-size int                   number of incoming events that can
be buffered (default 100)
      --eventd-workers int                       number of workers spawned for
processing incoming events (default 100)
  -h, --help                                     help for start
```

```
      --insecure-skip-tls-verify                    skip TLS verification (not
recommended!)
      --jwt-private-key-file string                 path to the PEM-encoded private key
to use to sign JWTs
      --jwt-public-key-file string                  path to the PEM-encoded public key
to use to verify JWT signatures
      --keepalived-buffer-size int                   number of incoming keepalives that
can be buffered (default 100)
      --keepalived-workers int                       number of workers spawned for
processing incoming keepalives (default 100)
      --key-file string                             TLS certificate key in PEM format
      --labels stringToString                        entity labels map (default [])
      --log-level string                             logging level [panic, fatal,
error, warn, info, debug, trace] (default "warn")
      --metrics-refresh-interval string             Go duration value (e.g. 1h5m30s)
that governs how often metrics are refreshed. (default "1m")
      --pipelined-buffer-size int                    number of events to handle that
can be buffered (default 100)
      --pipelined-workers int                        number of workers spawned for
handling events through the event pipeline (default 100)
      --platform-metrics-log-file string            platform metrics log file path
      --platform-metrics-logging-interval string    platform metrics logging interval
      --require-fips                                indicates whether fips support
should be required in openssl
      --trusted-ca-file string                      TLS CA certificate bundle in PEM
format

Store Flags:
      --etcd-advertise-client-urls strings          list of this member's client URLs
to advertise to clients (default [http://localhost:2379])
      --etcd-cert-file string                       path to the client server TLS cert
file
      --etcd-cipher-suites strings                   list of ciphers to use for etcd
TLS configuration
      --etcd-client-cert-auth                        enable client cert authentication
      --etcd-client-urls string                      client URLs to use when operating
as an etcd client
      --etcd-discovery string                        discovery URL used to bootstrap
the cluster
      --etcd-discovery-srv string                    DNS SRV record used to bootstrap
the cluster
      --etcd-election-timeout uint                   time in ms a follower node will go
```

```
without hearing a heartbeat before attempting to become leader itself (default 3000)
      --etcd-heartbeat-interval uint              interval in ms with which the etcd
leader will notify followers that it is still the leader (default 300)
      --etcd-initial-advertise-peer-urls strings  list of this member's peer URLs to
advertise to the rest of the cluster (default [http://127.0.0.1:2380])
      --etcd-initial-cluster string               initial cluster configuration for
bootstrapping
      --etcd-initial-cluster-state string         initial cluster state ("new" or
"existing") (default "new")
      --etcd-initial-cluster-token string         initial cluster token for the etcd
cluster during bootstrap
      --etcd-key-file string                      path to the client server TLS key
file
      --etcd-client-log-level string              etcd client logging level [panic,
fatal, error, warn, info, debug] (default "error")
      --etcd-listen-client-urls strings           list of etcd client URLs to listen
on (default [http://127.0.0.1:2379])
      --etcd-listen-peer-urls strings             list of URLs to listen on for peer
traffic (default [http://127.0.0.1:2380])
      --etcd-log-level string                     etcd server logging level [panic,
fatal, error, warn, info, debug]
      --etcd-max-request-bytes uint               maximum etcd request size in bytes
(use with caution) (default 1572864)
      --etcd-name string                          name for this etcd node (default
"default")
      --etcd-peer-cert-file string                path to the peer server TLS cert
file
      --etcd-peer-client-cert-auth                enable peer client cert
authentication
      --etcd-peer-key-file string                 path to the peer server TLS key file
      --etcd-peer-trusted-ca-file string          path to the peer server TLS trusted
CA file
      --etcd-quota-backend-bytes int              maximum etcd database size in
bytes (use with caution) (default 4294967296)
      --etcd-trusted-ca-file string               path to the client server TLS
trusted CA cert file
      --etcd-unsafe-no-fsync                      disables fsync, unsafe, may cause
data loss
      --no-embed-etcd                             don't embed etcd, use external
etcd instead
```

The backend requires that the `state-dir` configuration option is set before starting. All other required flags have default values.

For more information about log configuration options, read Event logging and Platform metrics logging.

General configuration

annotations	
description	Non-identifying metadata to include with entity data for backend dynamic runtime assets (for example, handler and mutator dynamic runtime assets). ***NOTE**: For annotations that you define in backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the annotation `webhookURL: "https://my-webhook.com"` in backend.yml, it will be listed as `webhookurl: "https://my-webhook.com"` in entity definitions.* *Key cases are **not** modified for annotations you define with the `--annotations` command line flag or the `SENSU_BACKEND_ANNOTATIONS` environment variable.*
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_BACKEND_ANNOTATIONS`
command line example	`sensu-backend start --annotations sensu.io/plugins/slack/config/webhook-url=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXX` `sensu-backend start --annotations example-key="example value" --annotations example-key2="example value"`

backend.yml config file example	<pre>annotations: sensu.io/plugins/slack/config/webhook-url: "https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXX"</pre>

api-listen-address

description	Address the API daemon will listen for requests on.
type	String
default	`[::]:8080`
environment variable	`SENSU_BACKEND_API_LISTEN_ADDRESS`
command line example	<pre>sensu-backend start --api-listen-address [::]:8080</pre>
backend.yml config file example	<pre>api-listen-address: "[::]:8080"</pre>

api-request-limit

description	Maximum size for API request bodies. In bytes.
type	Integer
default	`512000`
environment variable	`SENSU_BACKEND_API_REQUEST_LIMIT`
command line example	<pre>sensu-backend start --api-request-limit 1024000</pre>
backend.yml config	

file example	`api-request-limit: 1024000`

api-serve-wait-time

description	Time to wait after starting the backend before serving API requests. In seconds.
type	String
default	`0s`
environment variable	`SENSU_BACKEND_API_SERVE_WAIT_TIME`
command line example	`sensu-backend start --api-serve-wait-time 10s`
backend.yml config file example	`api-serve-wait-time: 10s`

api-url

description	URL used to connect to the API.
type	String
default	`http://localhost:8080` (Debian and RHEL families) `http://$SENSU_HOSTNAME:8080` (Docker) ***NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.*
environment variable	`SENSU_BACKEND_API_URL`
command line	`sensu-backend start --api-url http://localhost:8080`

example	
backend.yml config file example	`api-url: "http://localhost:8080"`

api-write-timeout

description	Maximum amount of time to wait before timing out on API HTTP server response writes. In milliseconds (`ms`), seconds (`s`), minutes (`m`), or hours (`h`).
type	String
default	`15s`
environment variable	`SENSU_BACKEND_API_WRITE_TIMEOUT`
command line example	`sensu-backend start --api-write-timeout 15s`
backend.yml config file example	`api-write-timeout: 15s`

assets-burst-limit

description	Maximum amount of burst allowed in a rate interval when fetching dynamic runtime assets.
type	Integer
default	`100`
environment variable	`SENSU_BACKEND_ASSETS_BURST_LIMIT`
command line example	`sensu-backend start --assets-burst-limit 100`

backend.yml config file example	`assets-burst-limit: 100`

assets-rate-limit

description	Maximum number of dynamic runtime assets to fetch per second. The default value `1.39` is equivalent to approximately 5000 user-to-server requests per hour.
type	Float
default	`1.39`
environment variable	`SENSU_BACKEND_ASSETS_RATE_LIMIT`
command line example	`sensu-backend start --assets-rate-limit 1.39`
backend.yml config file example	`assets-rate-limit: 1.39`

cache-dir

description	Path to store cached data.
type	String
default	`/var/cache/sensu/sensu-backend`
environment variable	`SENSU_BACKEND_CACHE_DIR`
command line example	`sensu-backend start --cache-dir /var/cache/sensu-backend`
backend.yml config	

file example	`cache-dir: "/var/cache/sensu-backend"`

config-file

description	Path to Sensu backend config file.
type	String
default	`/etc/sensu/backend.yml`
environment variable	`SENSU_BACKEND_CONFIG_FILE`
command line example	`sensu-backend start --config-file /etc/sensu/backend.yml` `sensu-backend start -c /etc/sensu/backend.yml`

debug

description	If `true`, enable debugging and profiling features for use with the Go pprof package. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_DEBUG`
command line example	`sensu-backend start --debug`
backend.yml config file example	`debug: true`

deregistration-handler

description	Name of the default event handler to use when processing agent deregistration events.
type	String
default	`""`
environment variable	`SENSU_BACKEND_DEREGISTRATION_HANDLER`
command line example	`sensu-backend start --deregistration-handler deregister`
backend.yml config file example	`deregistration-handler: "deregister"`

labels

description	Custom attributes to include with entity data for backend dynamic runtime assets (for example, handler and mutator dynamic runtime assets). ***NOTE**: For labels that you define in backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the label `securityZone: "us-west-2a"` in backend.yml, it will be listed as `securityzone: "us-west-2a"` in entity definitions.* *Key cases are **not** modified for labels you define with the `--labels` command line flag or the `SENSU_BACKEND_LABELS` environment variable.*
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

environment variable	`SENSU_BACKEND_LABELS`
command line example	`sensu-backend start --labels security_zone=us-west-2a` `sensu-backend start --labels example_key1="example value" example_key2="example value"`
backend.yml config file example	`labels:` `  security_zone: "us-west-2a"` `  example_key1: "example value"` `  example_key2: "example value"`

log-level

description	Logging level: `panic`, `fatal`, `error`, `warn`, `info`, `debug`, or `trace`.
type	String
default	`warn`
environment variable	`SENSU_BACKEND_LOG_LEVEL`
command line example	`sensu-backend start --log-level debug`
backend.yml config file example	`log-level: "debug"`

metrics-refresh-interval

description	Interval at which Sensu should refresh metrics. In hours, minutes, seconds, or a combination — for example, `5m`, `1m30s`, and `1h10m30s` are all valid values.

***COMMERCIAL FEATURE**: Access the `metrics-refresh-interval` configuration option in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

type	String
default	`1m`
environment variable	`SENSU_BACKEND_METRICS_REFRESH_INTERVAL`
command line example	`sensu-backend start --metrics-refresh-interval 10s`
backend.yml config file example	`metrics-refresh-interval: 10s`

state-dir

description	Path to Sensu state storage: `/var/lib/sensu/sensu-backend`.
type	String
required	true
environment variable	`SENSU_BACKEND_STATE_DIR`
command line example	`sensu-backend start --state-dir /var/lib/sensu/sensu-backend` `sensu-backend start -d /var/lib/sensu/sensu-backend`
backend.yml config file example	`state-dir: "/var/lib/sensu/sensu-backend"`

Agent communication configuration

agent-auth-cert-file

description	TLS certificate in PEM format for agent certificate authentication. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_CERT_FILE`
command line example	`sensu-backend start --agent-auth-cert-file /path/to/tls/backend-1.pem`
backend.yml config file example	`agent-auth-cert-file: /path/to/tls/backend-1.pem`

agent-auth-crl-urls

description	URLs of CRLs for agent certificate authentication. The Sensu backend uses this list to perform a revocation check for agent mTLS.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_CRL_URLS`
command line example	`sensu-backend start --agent-auth-crl-urls http://localhost/CARoot.crl`
backend.yml config	

file example	`agent-auth-crl-urls: http://localhost/CARoot.crl`

agent-auth-key-file

description	TLS certificate key in PEM format for agent certificate authentication.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_KEY_FILE`
command line example	`sensu-backend start --agent-auth-key-file /path/to/tls/backend-1-key.pem`
backend.yml config file example	`agent-auth-key-file: /path/to/tls/backend-1-key.pem`

agent-auth-trusted-ca-file

description	TLS CA certificate bundle in PEM format for agent certificate authentication.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_TRUSTED_CA_FILE`
command line example	`sensu-backend start --agent-auth-trusted-ca-file /path/to/tls/ca.pem`
backend.yml config file example	`agent-auth-trusted-ca-file: /path/to/tls/ca.pem`

agent-burst-limit

description	Maximum amount of burst allowed in a rate interval for agent transport WebSocket connections. ***NOTE**: The `agent-burst-limit` configuration flag is deprecated.* ***COMMERCIAL FEATURE**: Access the `agent-burst-limit` configuration option in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*
type	Integer
default	`null`
environment variable	`SENSU_BACKEND_AGENT_BURST_LIMIT`
command line example	`sensu-backend start --agent-burst-limit 10`
backend.yml config file example	`agent-burst-limit: 10`

agent-host

description	Agent listener host. Listens on all IPv4 and IPv6 addresses by default.
type	String
default	`[::]`
environment variable	`SENSU_BACKEND_AGENT_HOST`
command line	

example	`sensu-backend start --agent-host 127.0.0.1`
backend.yml config file example	`agent-host: "127.0.0.1"`

agent-serve-wait-time

description	Time to wait after starting the backend before accepting agent connections. In seconds.
type	String
default	`0s`
environment variable	`SENSU_BACKEND_AGENT_LISTEN_WAIT_TIME`
command line example	`sensu-backend start --agent-serve-wait-time 10s`
backend.yml config file example	`agent-serve-wait-time: 10s`

agent-port

description	Agent listener port.
type	Integer
default	`8081`
environment variable	`SENSU_BACKEND_AGENT_PORT`
command line example	`sensu-backend start --agent-port 8081`

backend.yml config file example	`agent-port: 8081`

agent-rate-limit

description	Maximum number of agent transport WebSocket connections per second, per backend. ***COMMERCIAL FEATURE**: Access the `agent-rate-limit` configuration option in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*
type	Integer
default	`null`
environment variable	`SENSU_BACKEND_AGENT_RATE_LIMIT`
command line example	`sensu-backend start --agent-rate-limit 10`
backend.yml config file example	`agent-rate-limit: 10`

Security configuration

cert-file

description	Path to the primary backend certificate file. Specifies a fallback SSL/TLS certificate if the `dashboard-cert-file` configuration option is not used. This certificate secures communications between the Sensu web UI and end user web browsers, as well as communication between sensuctl and the Sensu API. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.

type	String
default	`""`
environment variable	`SENSU_BACKEND_CERT_FILE`
command line example	`sensu-backend start --cert-file /path/to/tls/backend-1.pem`
backend.yml config file example	`cert-file: "/path/to/tls/backend-1.pem"`

insecure-skip-tls-verify

description

If `true`, skip SSL verification. Otherwise, `false`.

***WARNING**: This configuration option is intended for use in development systems only. Do not use this configuration option in production.*

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_INSECURE_SKIP_TLS_VERIFY`
command line example	`sensu-backend start --insecure-skip-tls-verify`
backend.yml config file example	`insecure-skip-tls-verify: true`

jwt-private-key-file

description	Path to the PEM-encoded private key to use to sign JSON Web Tokens (JWTs). ***NOTE**: The internal symmetric secret key is used by default to sign all JWTs unless a private key is specified via this attribute.*
type	String
default	`""`
environment variable	`SENSU_BACKEND_JWT_PRIVATE_KEY_FILE`
command line example	`sensu-backend start --jwt-private-key-file /path/to/key/private.pem`
backend.yml config file example	`jwt-private-key-file: /path/to/key/private.pem`

jwt-public-key-file

description	Path to the PEM-encoded public key to use to verify JSON Web Token (JWT) signatures. ***NOTE**: JWTs signed with the internal symmetric secret key will continue to be verified with that key.*
type	String
default	`""`
environment variable	`SENSU_BACKEND_JWT_PUBLIC_KEY_FILE`
required	false, unless `jwt-private-key-file` is defined
command line	

example	`sensu-backend start --jwt-public-key-file /path/to/key/public.pem`
backend.yml config file example	`jwt-public-key-file: /path/to/key/public.pem`

key-file

description	Path to the primary backend key file. Specifies a fallback SSL/TLS key if the `dashboard-key-file` configuration option is not used. This key secures communication between the Sensu web UI and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`
environment variable	`SENSU_BACKEND_KEY_FILE`
command line example	`sensu-backend start --key-file /path/to/tls/backend-1-key.pem`
backend.yml config file example	`key-file: "/path/to/tls/backend-1-key.pem"`

require-fips

description	Require Federal Information Processing Standard (FIPS) support in OpenSSL. Logs an error at Sensu backend startup if `true` but OpenSSL is not running in FIPS mode. ***NOTE**: The `require-fips` configuration option is only available within the Linux amd64 OpenSSL-linked binary. Contact Sensu to*

request the builds for OpenSSL with FIPS support.

type	Boolean
default	false
environment variable	`SENSU_BACKEND_REQUIRE_FIPS`
command line example	`sensu-backend start --require-fips`
backend.yml config file example	`require-fips: true`

require-openssl

description

Use OpenSSL instead of Go's standard cryptography library. Logs an error at Sensu backend startup if `true` but Go's standard cryptography library is loaded.

***NOTE**: The `require-openssl` configuration option is only available within the Linux amd64 OpenSSL-linked binary. Contact Sensu to request the builds for OpenSSL with FIPS support.*

type	Boolean
default	false
environment variable	`SENSU_BACKEND_REQUIRE_OPENSSL`
command line example	`sensu-backend start --require-openssl`
backend.yml config file example	`require-openssl: true`

trusted-ca-file	
description	Path to the primary backend CA file. Specifies a fallback SSL/TLS certificate authority in PEM format used for etcd client (mutual TLS) communication if the `etcd-trusted-ca-file` is not used. This CA file is used in communication between the Sensu web UI and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`
environment variable	`SENSU_BACKEND_TRUSTED_CA_FILE`
command line example	`sensu-backend start --trusted-ca-file /path/to/tls/ca.pem`
backend.yml config file example	`trusted-ca-file: "/path/to/tls/ca.pem"`

Web UI configuration

dashboard-cert-file	
description	Web UI TLS certificate in PEM format. This certificate secures communication with the Sensu web UI. If the `dashboard-cert-file` is not provided in the backend configuration, Sensu uses the certificate specified in the `cert-file` configuration option for the web UI. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.
type	String
default	`""`

environment variable	`SENSU_BACKEND_DASHBOARD_CERT_FILE`
command line example	`sensu-backend start --dashboard-cert-file /path/to/tls/separate-webui-cert.pem`
backend.yml config file example	`dashboard-cert-file: "/path/to/tls/separate-webui-cert.pem"`

dashboard-host

description	Web UI listener host.
type	String
default	`[::]`
environment variable	`SENSU_BACKEND_DASHBOARD_HOST`
command line example	`sensu-backend start --dashboard-host 127.0.0.1`
backend.yml config file example	`dashboard-host: "127.0.0.1"`

dashboard-key-file

description	Web UI TLS certificate key in PEM format. This key secures communication with the Sensu web UI. If the `dashboard-key-file` is not provided in the backend configuration, Sensu uses the key specified in the `key-file` configuration option for the web UI.
type	String
default	`""`

environment variable	`SENSU_BACKEND_DASHBOARD_KEY_FILE`
command line example	`sensu-backend start --dashboard-key-file /path/to/tls/separate-webui-key.pem`
backend.yml config file example	`dashboard-key-file: "/path/to/tls/separate-webui-key.pem"`

dashboard-port

description	Web UI listener port.
type	Integer
default	`3000`
environment variable	`SENSU_BACKEND_DASHBOARD_PORT`
command line example	`sensu-backend start --dashboard-port 3000`
backend.yml config file example	`dashboard-port: 3000`

dashboard-write-timeout

description	Maximum amount of time to wait before timing out on web UI HTTP server response writes. In milliseconds (`ms`), seconds (`s`), minutes (`m`), or hours (`h`).
type	String
default	`15s`
environment variable	`SENSU_BACKEND_DASHBOARD_WRITE_TIMEOUT`

command line example	`sensu-backend start --dashboard-write-timeout 15s`
backend.yml config file example	`dashboard-write-timeout: 15s`

Datastore and cluster configuration

etcd-advertise-client-urls

description	List of this member's client URLs to advertise to the rest of the cluster. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	List
default	`http://localhost:2379` (Debian and RHEL families) `http://$SENSU_HOSTNAME:2379` (Docker) ***NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.*
environment variable	`SENSU_BACKEND_ETCD_ADVERTISE_CLIENT_URLS`
command line example	`sensu-backend start --etcd-advertise-client-urls http://localhost:2378,http://localhost:2379` `sensu-backend start --etcd-advertise-client-urls`

```
http://localhost:2378 --etcd-advertise-client-urls
http://localhost:2379
```

backend.yml config file example	`etcd-advertise-client-urls:` `  - http://localhost:2378` `  - http://localhost:2379`

etcd-cert-file

description	Path to the etcd client API TLS certificate file. Secures communication between the embedded etcd client API and any etcd clients. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_CERT_FILE`
command line example	`sensu-backend start --etcd-cert-file /path/to/tls/backend-1.pem`
backend.yml config file example	`etcd-cert-file: "/path/to/tls/backend-1.pem"`

etcd-cipher-suites

description	List of allowed cipher suites for etcd TLS configuration. Sensu supports TLS 1.0-1.2 cipher suites as listed in the Go TLS documentation. You can use this attribute to defend your TLS servers from attacks on weak TLS ciphers. Go determines the default cipher suites based on the hardware used. ***NOTE**: To use TLS 1.3, add the following environment variable: `GODEBUG="tls13=1"`.* *To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
recommended	`etcd-cipher-suites:` `  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` `  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` `  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` `  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` `  - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305` `  - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305`
type	List
environment variable	`SENSU_BACKEND_ETCD_CIPHER_SUITES`
command line example	`sensu-backend start --etcd-cipher-suites TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` `sensu-backend start --etcd-cipher-suites TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 --etcd-cipher-suites TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
backend.yml config file example	`etcd-cipher-suites:` `  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`

```
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

etcd-client-cert-auth

description

If `true`, enable client certificate authentication. Otherwise, `false`.

> ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_ETCD_CLIENT_CERT_AUTH`
command line example	`sensu-backend start --etcd-client-cert-auth`
backend.yml config file example	`etcd-client-cert-auth: true`

etcd-client-log-level

description

Logging level for the internal etcd client: `panic`, `fatal`, `error`, `warn`, `info`, or `debug`.

type	String
default	`error`
environment variable	`SENSU_BACKEND_ETCD_CLIENT_LOG_LEVEL`
command line	`sensu-backend start --etcd-client-log-level error`

example	
backend.yml config file example	`etcd-client-log-level: "error"`

etcd-client-urls

description	List of client URLs to use when a sensu-backend is not operating as an etcd member. To configure sensu-backend for use with an external etcd instance, use this configuration option in conjunction with `no-embed-etcd` when executing sensu-backend start or sensu-backend init. If you do not use this option when using `no-embed-etcd`, sensu-backend start and sensu-backend-init will fall back to –etcd-listen-client-urls. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	List
default	`http://127.0.0.1:2379`
environment variable	`SENSU_BACKEND_ETCD_CLIENT_URLS`
command line example	`sensu-backend start --etcd-client-urls 'https://10.0.0.1:2379 https://10.1.0.1:2379'` `sensu-backend start --etcd-client-urls https://10.0.0.1:2379 --etcd-client-urls https://10.1.0.1:2379`
backend.yml config file example	`etcd-client-urls:` `  - https://10.0.0.1:2379` `  - https://10.1.0.1:2379`

etcd-discovery

description	Exposes etcd's embedded auto-discovery features. Attempts to use etcd discovery to get the cluster configuration. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	""
environment variable	`SENSU_BACKEND_ETCD_DISCOVERY`
command line example	`sensu-backend start --etcd-discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de`
backend.yml config file example	`etcd-discovery:` `  - https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de`

etcd-discovery-srv

description	Exposes etcd's embedded auto-discovery features. Attempts to use a DNS SRV record to get the cluster configuration. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend*

command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	String
default	""
environment variable	`SENSU_BACKEND_ETCD_DISCOVERY_SRV`
command line example	`sensu-backend start --etcd-discovery-srv example.org`
backend.yml config file example	`etcd-discovery-srv:` `  - example.org`

etcd-initial-advertise-peer-urls

description	List of this member's peer URLs to advertise to the rest of the cluster. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	List
default	`http://127.0.0.1:2380` (Debian and RHEL families) `http://$SENSU_HOSTNAME:2380` (Docker) ***NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.*

environment variable	`SENSU_BACKEND_ETCD_INITIAL_ADVERTISE_PEER_URLS`
command line example	`sensu-backend start --etcd-initial-advertise-peer-urls https://10.0.0.1:2380,https://10.1.0.1:2380` `sensu-backend start --etcd-initial-advertise-peer-urls https://10.0.0.1:2380 --etcd-initial-advertise-peer-urls https://10.1.0.1:2380`
backend.yml config file example	`etcd-initial-advertise-peer-urls:` `  - https://10.0.0.1:2380` `  - https://10.1.0.1:2380`

etcd-initial-cluster

description	Initial cluster configuration for bootstrapping. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	`default=http://127.0.0.1:2380` (Debian and RHEL families) `default=http://$SENSU_HOSTNAME:2380` (Docker) ***NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.*
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER`

command line example	`sensu-backend start --etcd-initial-cluster backend-0=https://10.0.0.1:2380,backend-1=https://10.1.0.1:2380,backend-2=https://10.2.0.1:2380`
backend.yml config file example	`etcd-initial-cluster: "backend-0=https://10.0.0.1:2380,backend-1=https://10.1.0.1:2380,backend-2=https://10.2.0.1:2380"`

etcd-initial-cluster-state

description	Initial cluster state (`new` or `existing`). ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	`new`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER_STATE`
command line example	`sensu-backend start --etcd-initial-cluster-state existing`
backend.yml config file example	`etcd-initial-cluster-state: "existing"`

etcd-initial-cluster-token

description	Unique token for the etcd cluster. Provide the same `etcd-initial-cluster-token` value for each cluster member. The `etcd-initial-cluster-token` allows etcd to generate unique cluster IDs and member IDs even for clusters with otherwise identical configurations, which prevents cross-cluster-interaction and potential cluster corruption. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER_TOKEN`
command line example	`sensu-backend start --etcd-initial-cluster-token unique_token_for_this_cluster`
backend.yml config file example	`etcd-initial-cluster-token: "unique_token_for_this_cluster"`

etcd-key-file

description	Path to the etcd client API TLS key file. Secures communication between the embedded etcd client API and any etcd clients. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	String
environment variable	`SENSU_BACKEND_ETCD_KEY_FILE`
command line example	`sensu-backend start --etcd-key-file /path/to/tls/backend-1-key.pem`
backend.yml config file example	`etcd-key-file: "/path/to/tls/backend-1-key.pem"`

etcd-listen-client-urls

description

List of URLs to listen on for client traffic. Sensu's default embedded etcd configuration listens for unencrypted client communication on port 2379.

***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	List
default	`http://127.0.0.1:2379` (Debian and RHEL families) `http://[::]:2379` (Docker)
environment variable	`SENSU_BACKEND_ETCD_LISTEN_CLIENT_URLS`
command line example	`sensu-backend start --etcd-listen-client-urls https://10.0.0.1:2379,https://10.1.0.1:2379` `sensu-backend start --etcd-listen-client-urls https://10.0.0.1:2379 --etcd-listen-client-urls https://10.1.0.1:2379`

backend.yml config file example	`etcd-listen-client-urls:` `  - https://10.0.0.1:2379` `  - https://10.1.0.1:2379`

etcd-listen-peer-urls

description	List of URLs to listen on for peer traffic. Sensu's default embedded etcd configuration listens for unencrypted peer communication on port 2380. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	List
default	`http://127.0.0.1:2380` (Debian and RHEL families) `http://[::]:2380` (Docker)
environment variable	`SENSU_BACKEND_ETCD_LISTEN_PEER_URLS`
command line example	`sensu-backend start --etcd-listen-peer-urls https://10.0.0.1:2380,https://10.1.0.1:2380` `sensu-backend start --etcd-listen-peer-urls https://10.0.0.1:2380 --etcd-listen-peer-urls https://10.1.0.1:2380`
backend.yml config file example	`etcd-listen-peer-urls:` `  - https://10.0.0.1:2380` `  - https://10.1.0.1:2380`

etcd-log-level

description	Logging level for the embedded etcd server: `panic`, `fatal`, `error`, `warn`, `info`, or `debug`. Defaults to value provided for the backend log level. If the backend log level is set to `trace`, the etcd log level will be set to `debug` (`trace` is not a valid etcd log level).
type	String
default	Backend log level value (or `debug`, if the backend log level is set to `trace`)
environment variable	`SENSU_BACKEND_ETCD_LOG_LEVEL`
command line example	`sensu-backend start --etcd-log-level debug`
backend.yml config file example	`etcd-log-level: "debug"`

etcd-name

description	Human-readable name for this member. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	`default`
environment variable	`SENSU_BACKEND_ETCD_NAME`
command line example	`sensu-backend start --etcd-name backend-0`

backend.yml config file example	`etcd-name: "backend-0"`

etcd-peer-cert-file

description	Path to the peer server TLS certificate file. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_CERT_FILE`
command line example	`sensu-backend start --etcd-peer-cert-file /path/to/tls/backend-1.pem`
backend.yml config file example	`etcd-peer-cert-file: "/path/to/tls/backend-1.pem"`

etcd-peer-client-cert-auth

description	Enable peer client certificate authentication. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_ETCD_PEER_CLIENT_CERT_AUTH`
command line example	`sensu-backend start --etcd-peer-client-cert-auth`
backend.yml config file example	`etcd-peer-client-cert-auth: true`

etcd-peer-key-file

description

Path to the etcd peer API TLS key file. Secures communication between etcd cluster members.

***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_KEY_FILE`
command line example	`sensu-backend start --etcd-peer-key-file /path/to/tls/backend-1-key.pem`
backend.yml config file example	`etcd-peer-key-file: "/path/to/tls/backend-1-key.pem"`

etcd-peer-trusted-ca-file

description	Path to the etcd peer API server TLS trusted CA file. Secures communication between etcd cluster members. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_TRUSTED_CA_FILE`
command line example	`sensu-backend start --etcd-peer-trusted-ca-file ./ca.pem`
backend.yml config file example	`etcd-peer-trusted-ca-file: "./ca.pem"`

etcd-trusted-ca-file

description	Path to the client server TLS trusted CA certificate file. Secures communication with the etcd client server. ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	String
default	`""`

environment variable	`SENSU_BACKEND_ETCD_TRUSTED_CA_FILE`
command line example	`sensu-backend start --etcd-trusted-ca-file ./ca.pem`
backend.yml config file example	`etcd-trusted-ca-file: "./ca.pem"`

etcd-unsafe-no-fsync

description	The `etcd-unsafe-no-fsync` configuration option allows you to run sensu-backend with an embedded etcd node for testing and development with less load on the file system. If `true`, disable fsync. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_ETCD_UNSAFE_NO_FSYNC`
command line example	`sensu-backend start --etcd-unsafe-no-fsync`
backend.yml config file example	`etcd-unsafe-no-fsync: true`

no-embed-etcd

description	If `true`, do not embed etcd (use external etcd instead). Otherwise, `false`.

***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_NO_EMBED_ETCD`
command line example	`sensu-backend start --no-embed-etcd`
backend.yml config file example	`no-embed-etcd: true`

Advanced configuration options

etcd-election-timeout

description	Time that a follower node will go without hearing a heartbeat before attempting to become leader itself. In milliseconds (ms). Set to at least 10 times the etcd-heartbeat-interval. Read the etcd time parameter documentation for details and other considerations.

***WARNING**: Make sure to set the same election timeout value for all etcd members in one cluster. Setting different values for etcd members may reduce cluster stability.*

***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	Integer
default	`3000`

environment variable	`SENSU_BACKEND_ETCD_ELECTION_TIMEOUT`
command line example	`sensu-backend start --etcd-election-timeout 3000`
backend.yml config file example	`etcd-election-timeout: 3000`

etcd-heartbeat-interval

description	Interval at which the etcd leader will notify followers that it is still the leader. In milliseconds (ms). Best practice is to set the interval based on round-trip time between members. Read the etcd time parameter documentation for details and other considerations. ***WARNING**: Make sure to set the same heartbeat interval value for all etcd members in one cluster. Setting different values for etcd members may reduce cluster stability.* ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	Integer
default	`300`
environment variable	`SENSU_BACKEND_ETCD_HEARTBEAT_INTERVAL`
command line example	`sensu-backend start --etcd-heartbeat-interval 300`
backend.yml config file example	`etcd-heartbeat-interval: 300`

etcd-max-request-bytes

description	Maximum etcd request size in bytes that can be sent to an etcd server by a client. Increasing this value allows etcd to process events with large outputs at the cost of overall latency. ***WARNING**: Use with caution. This configuration option requires familiarity with etcd. Improper use of this option can result in a non-functioning Sensu instance.* ***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*
type	Integer
default	`1572864`
environment variable	`SENSU_BACKEND_ETCD_MAX_REQUEST_BYTES`
command line example	`sensu-backend start --etcd-max-request-bytes 1572864`
backend.yml config file example	`etcd-max-request-bytes: 1572864`

etcd-quota-backend-bytes

description	Maximum etcd database size in bytes. Increasing this value allows for a larger etcd database at the cost of performance.

***WARNING**: Use with caution. This configuration option requires familiarity with etcd. Improper use of this option can result in a non-functioning Sensu instance.*

***NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	Integer
default	`4294967296`
environment variable	`SENSU_BACKEND_ETCD_QUOTA_BACKEND_BYTES`
command line example	`sensu-backend start --etcd-quota-backend-bytes 4294967296`
backend.yml config file example	`etcd-quota-backend-bytes: 4294967296`

eventd-buffer-size

description

Number of incoming events that can be buffered before being processed by an eventd worker.

***WARNING**: Modify with caution. Increasing this value may result in greater memory usage.*

type	Integer
default	`100`
environment variable	`SENSU_BACKEND_EVENTD_BUFFER_SIZE`

command line example	`sensu-backend start --eventd-buffer-size 100`
backend.yml config file example	`eventd-buffer-size: 100`

eventd-workers

description	Number of workers spawned for processing incoming events that are stored in the eventd buffer. ***WARNING**: Modify with caution. Increasing this value may result in greater CPU usage.*
type	Integer
default	`100`
environment variable	`SENSU_BACKEND_EVENTD_WORKERS`
command line example	`sensu-backend start --eventd-workers 100`
backend.yml config file example	`eventd-workers: 100`

keepalived-buffer-size

description	Number of incoming keepalives that can be buffered before being processed by a keepalived worker. ***WARNING**: Modify with caution. Increasing this value may result in greater memory usage.*

type	Integer
default	`100`
environment variable	`SENSU_BACKEND_KEEPALIVED_BUFFER_SIZE`
command line example	`sensu-backend start --keepalived-buffer-size 100`
backend.yml config file example	`keepalived-buffer-size: 100`

keepalived-workers

description

Number of workers spawned for processing incoming keepalives that are stored in the keepalived buffer.

***WARNING**: Modify with caution. Increasing this value may result in greater CPU usage.*

type	Integer
default	`100`
environment variable	`SENSU_BACKEND_KEEPALIVED_WORKERS`
command line example	`sensu-backend start --keepalived-workers 100`
backend.yml config file example	`keepalived-workers: 100`

pipelined-buffer-size

description	Number of events to handle that can be buffered before being processed by a pipelined worker. ***WARNING**: Modify with caution. Increasing this value may result in greater memory usage.*
type	Integer
default	`100`
environment variable	`SENSU_BACKEND_PIPELINED_BUFFER_SIZE`
command line example	`sensu-backend start --pipelined-buffer-size 100`
backend.yml config file example	`pipelined-buffer-size: 100`

pipelined-workers

description	Number of workers spawned for handling events through the event pipeline that are stored in the pipelined buffer. ***WARNING**: Modify with caution. Increasing this value may result in greater CPU usage.*
type	Integer
default	`100`
environment variable	`SENSU_BACKEND_PIPELINED_WORKERS`
command line example	`sensu-backend start --pipelined-workers 100`

backend.yml config file example	`pipelined-workers: 100`

Backend configuration methods

Backend configuration file

You can customize the backend configuration in a `.yml` configuration file. The default backend configuration file path for Linux is `/etc/sensu/backend.yml`.

To use the `backend.yml` file to configure the backend, list the desired configuration attributes and values. Review the example Sensu backend configuration file for a complete example.

> ***NOTE**: The backend loads configuration upon startup. If you make changes in the `backend.yml` configuration file after startup, you must restart the backend for the changes to take effect.*

Configuration via command line flags or environment variables overrides any configuration specified in the backend configuration file. Read Create overrides to learn more.

Command line flags

You can customize the backend configuration with `sensu-agent start` command line flags.

To use command line flags, specify the desired configuration options and values along with the `sensu-backend start` command. For example:

```
sensu-backend start --deregistration-handler slack_deregister --log-level debug
```

Configuration via command line flags overrides attributes specified in a configuration file or with environment variables. Read Create overrides to learn more.

Environment variables

Instead of using a configuration file or command line flags, you can use environment variables to configure your Sensu backend. Each backend configuration option has an associated environment variable. You can also create your own environment variables, as long as you name them correctly and save them in the correct place. Here's how.

1. Create the files from which the `sensu-backend` service configured by our supported packages will read environment variables:

   SHELL

   ```
   sudo touch /etc/default/sensu-backend
   ```

   SHELL

   ```
   sudo touch /etc/sysconfig/sensu-backend
   ```

2. Make sure the environment variable is named correctly.

   - To rename a configuration option you wish to specify as an environment variable, prepend `SENSU_BACKEND_`, convert dashes to underscores, and capitalize all letters. For example, the environment variable for the configuration option `api-listen-address` is `SENSU_BACKEND_API_LISTEN_ADDRESS`.

   - For a custom environment variable, you do not have to prepend `SENSU_BACKEND`. For example, `TEST_VAR_1` is a valid custom environment variable name.

3. Add the environment variable to the environment file.

   For example, to create `api-listen-address` as an environment variable and set it to `192.168.100.20:8080`:

   SHELL

   ```
   echo 'SENSU_BACKEND_API_LISTEN_ADDRESS=192.168.100.20:8080' | sudo tee -a /etc/default/sensu-backend
   ```

   SHELL

   ```
   ```

```
echo 'SENSU_BACKEND_API_LISTEN_ADDRESS=192.168.100.20:8080' | sudo tee -a
/etc/sysconfig/sensu-backend
```

4. Restart the sensu-backend service so these settings can take effect:

SHELL

```
sudo systemctl restart sensu-backend
```

SHELL

```
sudo systemctl restart sensu-backend
```

***NOTE**: Sensu includes an environment variable for each backend configuration option. They are listed in the configuration description tables.*

Format for label and annotation environment variables

To use labels and annotations as environment variables in your handler configurations, you must use a specific format when you create the label and annotation environment variables.

For example, to create the labels `"region": "us-east-1"` and `"type": "website"` as an environment variable:

SHELL

```
echo 'BACKEND_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo tee -a
/etc/default/sensu-backend
```

SHELL

```
echo 'BACKEND_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo tee -a
/etc/sysconfig/sensu-backend
```

To create the annotations `"maintainer": "Team A"` and `"webhook-url":`

`"https://hooks.slack.com/services/T0000/B00000/XXXXX"` as an environment variable:

SHELL

```
echo 'BACKEND_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/default/sensu-backend
```

SHELL

```
echo 'BACKEND_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/sysconfig/sensu-backend
```

Use environment variables with the Sensu backend

Any environment variables you create in `/etc/default/sensu-backend` (Debian family) or `/etc/sysconfig/sensu-backend` (RHEL family) will be available to handlers executed by the Sensu backend.

For example, if you create a custom environment variable `TEST_VARIABLE` in your sensu-backend file, it will be available to use in your handler configurations as `$TEST_VARIABLE`. The following handler will print the `TEST_VARIABLE` value set in your sensu-backend file in `/tmp/test.txt`:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: print_test_var
spec:
  command: echo $TEST_VARIABLE >> ./tmp/test.txt
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
```

```
  "api_version": "core/v2",
  "metadata": {
    "name": "print_test_var"
  },
  "spec": {
    "command": "echo $TEST_VARIABLE >> ./tmp/test.txt",
    "timeout": 0,
    "type": "pipe"
  }
}
```

***NOTE**: We recommend using secrets with the `Env` provider to expose secrets from environment variables on your Sensu backend nodes rather than using environment variables directly in your handler commands. Read the secrets reference and Use Env for secrets management for details.*

Create configuration overrides

Sensu has default settings and limits for certain configuration attributes, like the default log level. Depending on your environment and preferences, you may want to create overrides for these Sensu-specific defaults and limits.

You can create configuration overrides in several ways:

- Command line configuration flag arguments for `sensu-backend start`.
- Environment variables in `/etc/default/sensu-backend` (Debian family) or `/etc/sysconfig/sensu-backend` (RHEL family).
- Configuration settings in the backend.yml config file.

***NOTE**: We do not recommend editing the systemd unit file to create overrides. Future package upgrades can overwrite changes in the systemd unit file.*

Sensu applies the following precedence to override settings:

1. Arguments passed to the Sensu backend via command line configuration flags.
2. Environment variables in `/etc/default/sensu-backend` (Debian family) or

`/etc/sysconfig/sensu-backend` (RHEL family).
3. Configuration in the backend.yml config file.

For example, if you create overrides using all three methods, the command line configuration flag values will take precedence over the values you specify in `/etc/default/sensu-backend` or `/etc/sysconfig/sensu-backend` or the backend.yml config file.

Example override: Log level

The default log level for the Sensu backend is `warn`. To override the default and automatically apply a different log level for the backend, add the `--log-level` command line configuration flag when you start the Sensu backend. For example, to specify `debug` as the log level:

```
sensu-backend start --log-level debug
```

To configure an environment variable for the desired backend log level:

SHELL

```
echo 'SENSU_BACKEND_LOG_LEVEL=debug' | sudo tee -a /etc/default/sensu-backend
```

SHELL

```
echo 'SENSU_BACKEND_LOG_LEVEL=debug' | sudo tee -a /etc/sysconfig/sensu-backend
```

To configure the desired log level in the config file, add this line to backend.yml:

```
log-level: debug
```

Event logging

If you wish, you can log all Sensu event data to a file in JSON format. The Sensu event log file can be a reliable input source for your favorite data lake solution as well as a buffer for event data that you send to a database in case the database is unavailable.

***NOTE**: Event logs do not include log messages produced by sensu-backend service. To write Sensu service logs to flat files on disk, read Log Sensu services with systemd.*

Depending on the number and size of events, logging status and metrics events to a file can require intensive input/output (I/O) performance. Make sure you have adequate I/O capacity before using the event logging function.

***PRO TIP**: TCP stream handlers, which send observability event data to TCP sockets for external services to consume, are also a reliable way to transmit status and metrics event data without writing events to a local file.*

Use these backend configuration options to customize event logging:

event-log-buffer-size	
description	Buffer size of the event logger. Corresponds to the maximum number of events kept in memory in case the log file is temporarily unavailable or more events have been received than can be written to the log file.
type	Integer
default	100000
environment variable	`SENSU_BACKEND_EVENT_LOG_BUFFER_SIZE`
command line example	`sensu-backend start --event-log-buffer-size 100000`
backend.yml config file example	`event-log-buffer-size: 100000`

event-log-buffer-wait	
description	Buffer wait time for the event logger. When the buffer is full, the event logger will wait for the specified time for the writer to consume events

	from the buffer.
type	String
default	10ms
environment variable	`SENSU_BACKEND_EVENT_LOG_BUFFER_WAIT`
command line example	`sensu-backend start --event-log-buffer-wait 10ms`
backend.yml config file example	`event-log-buffer-wait: 10ms`

event-log-file

description	Path to the event log file. ***WARNING**: The log file should be located on a local drive. Logging directly to network drives is not supported.*
type	String
environment variable	`SENSU_BACKEND_EVENT_LOG_FILE`
command line example	`sensu-backend start --event-log-file /var/log/sensu/events.log`
backend.yml config file example	`event-log-file: "/var/log/sensu/events.log"`

event-log-parallel-encoders

description	Indicates whether Sensu should use parallel JSON encoders for event logging. If `true`, Sensu sets the number of JSON encoder workers to 50% of the total number of cores, with a minimum of 2 (for example, 6 JSON encoders on a 12-core machine). Otherwise, Sensu uses the default setting, which is a single JSON encoding worker. The `event-log-parallel-encoders` setting will not take effect unless you also specify a path to the event log file with the `event-log-file` configuration attribute.
type	Boolean
default	false
environment variable	`SENSU_BACKEND_EVENT_LOG_PARALLEL_ENCODERS`
command line example	`sensu-backend start --event-log-parallel-encoders true`
backend.yml config file example	`event-log-parallel-encoders: true`

Log rotation

To manually rotate event logs, first rename (move) the current log file. Then, send the *SIGHUP* signal to the sensu-backend process so it creates a new log file and starts logging to it. Most Linux distributions include `logrotate` to automatically rotate log files as a standard utility, configured to run once per day by default.

Because event log files can grow quickly for larger Sensu installations, we recommend using `logrotate` to automatically rotate log files more frequently. To use the example log rotation configurations listed below, you may need to configure `logrotate` to run once per hour.

Log rotation for systemd

In this example, the `postrotate` script will reload the backend after log rotate is complete.

```
/var/log/sensu/events.log
{
  rotate 3
  hourly
  missingok
  notifempty
  compress
  postrotate
    /bin/systemctl reload sensu-backend.service > /dev/null 2>/dev/null || true
  endscript
}
```

Without the `postrotate` script, the backend will not reload. This will cause sensu-backend (and sensu-agent, if translated for the Sensu agent) to no longer write to the log file, even if logrotate recreates the log file.

Log rotation for sysvinit

```
/var/log/sensu/events.log
{
  rotate 3
  hourly
  missingok
  notifempty
  compress
  postrotate
    kill -HUP `cat /var/run/sensu/sensu-backend.pid 2> /dev/null` 2> /dev/null ||
true
  endscript
}
```

Platform metrics logging

Sensu automatically writes core platform metrics in InfluxDB Line Protocol to a file at `/var/log/sensu/backend-stats.log`. You can use this file as an input source for your favorite data lake solution.

Metrics logging is enabled by default but can be disabled with the `disable-platform-metrics` configuration option.Sensu appends updated metrics at the interval you specify with the `platform-metrics-logging-interval` configuration option (default is every 60 seconds).

To rotate the platform metrics log, use the same methods as for event log rotation.

Use these backend configuration options to customize platform metrics logging:

disable-platform-metrics	
description	`true` to disable platform metrics logging. Otherwise, `false`.
type	Boolean
default	false
environment variable	`SENSU_BACKEND_DISABLE_PLATFORM_METRICS`
command line example	`sensu-backend start --disable-platform-metrics false`
backend.yml config file example	`disable-platform-metrics: false`

platform-metrics-log-file	
description	Path to the platform metrics log file. ***WARNING**: The log file should be located on a local drive. Logging directly to network drives is not supported.*
type	String
default	/var/log/sensu/sensu-backend/stats.log
environment variable	`SENSU_BACKEND_PLATFORM_METRICS_LOG_FILE`

command line example	`sensu-backend start --platform-metrics-log-file /var/log/sensu/sensu-backend/stats.log`
backend.yml config file example	`platform-metrics-log-file: "/var/log/sensu/sensu-backend/stats.log"`

platform-metrics-logging-interval	
description	Interval at which Sensu should append metrics to the platform metrics log.
type	String
default	60s
environment variable	`SENSU_BACKEND_PLATFORM_METRICS_LOGGING_INTERVAL`
command line example	`sensu-backend start --platform-metrics-logging-interval 60s`
backend.yml config file example	`platform-metrics-logging-interval: 60s`

Service management

***NOTE**: Service management commands may require administrative privileges.*

Start the service

Use the `sensu-backend` tool to start the backend and apply configuration flags.

Start the backend with configuration flags:

```
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

View available configuration flags and defaults:

```
sensu-backend start --help
```

If you do not include any configuration flags with the `sensu-backend start` command, the backend loads configuration from `/etc/sensu/backend.yml` by default.

Start the backend using a service manager:

```
sudo systemctl start sensu-backend
```

Stop the service

Stop the backend service using a service manager:

```
sudo systemctl stop sensu-backend
```

Restart the service

Restart the backend using a service manager:

```
sudo systemctl restart sensu-backend
```

You must restart the backend to implement any configuration updates.

Enable on boot

Enable the backend to start on system boot:

```
sudo systemctl enable sensu-backend
```

Disable the backend from starting on system boot:

```
sudo systemctl disable sensu-backend
```

***NOTE**: On older distributions of Linux, use `sudo chkconfig sensu-server on` to enable the backend and `sudo chkconfig sensu-server off` to disable the backend.*

Get service status

View the status of the backend service using a service manager:

```
sudo systemctl status sensu-backend
```

Get service version

Get the current backend version using the `sensu-backend` tool:

```
sensu-backend version
```

Get help

The `sensu-backend` tool provides general and command-specific help flags.

View sensu-backend commands:

```
sensu-backend help
```

List options for a specific command (in this case, sensu-backend start):

```
sensu-backend start --help
```

Checks reference

Checks work with Sensu agents to produce observability events automatically. You can use checks to monitor server resources, services, and application health as well as collect and analyze metrics. Read Monitor server resources to get started. Use Bonsai, the Sensu asset hub, to discover, download, and share Sensu check dynamic runtime assets.

Check example (minimum recommended attributes)

This example shows a check resource definition that includes the minimum recommended attributes.

> ***NOTE**: The attribute `interval` is not required if a valid `cron` schedule is defined. Read scheduling for more information.*

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_minimum
spec:
  command: collect.sh
  handlers:
  - slack
  interval: 10
  publish: true
  subscriptions:
  - system
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "check_minimum"
  },
  "spec": {
    "command": "collect.sh",
    "subscriptions": [
      "system"
    ],
    "handlers": [
      "slack"
    ],
    "interval": 10,
    "publish": true
  }
}
```

Check commands

Each Sensu check definition specifies a command and the schedule at which it should be executed. Check commands are executable commands that the Sensu agent executes.

A command may include command line arguments for controlling the behavior of the command executable. Many common checks are available as dynamic runtime assets from Bonsai and support command line arguments so different check definitions can use the same executable.

> ***NOTE**: Sensu advises against requiring root privileges to execute check commands or scripts. The Sensu user is not permitted to kill timed-out processes invoked by the root user, which could result in zombie processes.*

Check command execution

All check commands are executed by Sensu agents as the `sensu` user. Commands must be executable files that are discoverable on the Sensu agent system (for example, installed in a system `$PATH` directory).

Check result specification

Although Sensu agents attempt to execute any command defined for a check, successful check result processing requires adherence to a simple specification.

- Result data is output to stdout or stderr.
  - For service checks, this output is typically a human-readable message.
  - For metric checks, this output contains the measurements gathered by the check.
- Exit status code indicates state.
  - `0` indicates OK.
  - `1` indicates WARNING.
  - `2` indicates CRITICAL.
  - Exit status codes other than `0`, `1`, and `2` indicate an UNKNOWN or custom status

> ***PRO TIP**: If you're familiar with the **Nagios** monitoring system, you may recognize this specification — it is the same one that Nagios plugins use. As a result, you can use Nagios plugins with Sensu without any modification.*

At every execution of a check command, regardless of success or failure, the Sensu agent publishes the check's result for eventual handling by the **event processor** (the Sensu backend).

Check scheduling

The Sensu backend schedules checks and publishes check execution requests to entities via a publish/subscribe model. Checks have a defined set of subscriptions: transport topics to which the Sensu backend publishes check requests. Sensu entities become subscribers to these topics (called subscriptions) via their individual `subscriptions` attribute.

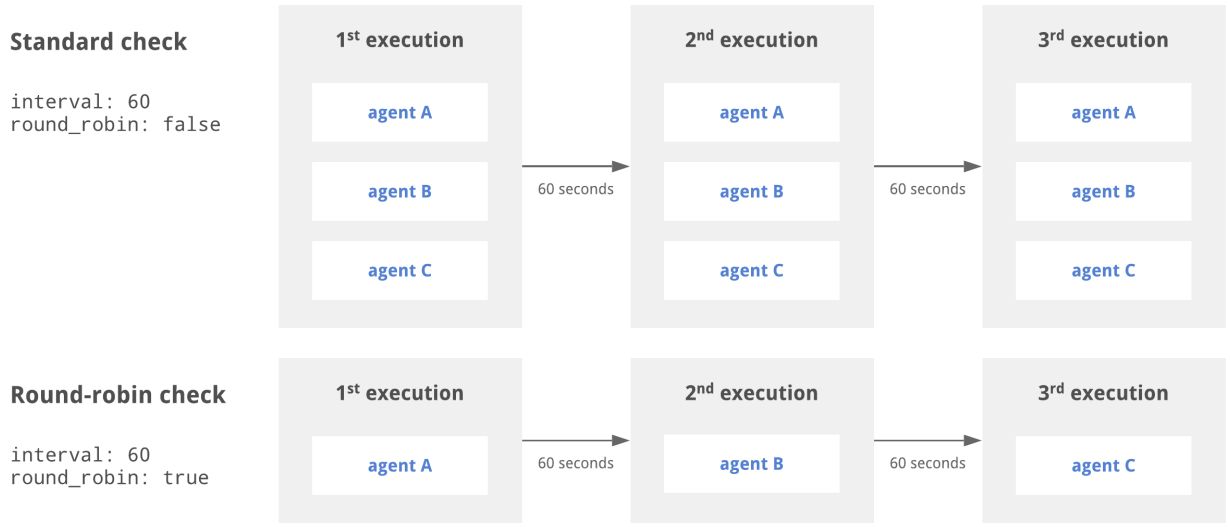
You can schedule checks using the `interval`, `cron`, and `publish` attributes. Sensu requires that checks include either an `interval` attribute (interval scheduling) or a `cron` attribute (cron scheduling).

Round robin checks

By default, Sensu schedules checks once per interval for each agent with a matching subscription: one check execution per agent per interval. Sensu also supports deduplicated check execution when configured with the `round_robin` check attribute. For checks with `round_robin` set to `true`, Sensu

executes the check once per interval, cycling through the available agents alphabetically according to agent name.

For example, for three agents configured with the `system` subscription (agents A, B, and C), a check configured with the `system` subscription and `round_robin` set to `true` results in one observability event per interval, with the agent creating the event following the pattern A -> B -> C -> A -> B -> C for the first six intervals.



In the diagram above, the standard check is executed by agents A, B, and C every 60 seconds. The round robin check cycles through the available agents, resulting in each agent executing the check every 180 seconds.

To use check `ttl` and `round_robin` together, your check configuration must also specify a `proxy_entity_name`. If you do not specify a `proxy_entity_name` when using check `ttl` and `round_robin` together, your check will stop executing.

> ***PRO TIP**: Use round robin to distribute check execution workload across multiple agents when using proxy checks.*

Event storage for round robin scheduling

If you use round robin scheduling for check execution, we recommend using PostgreSQL rather than etcd for event storage. Etcd leases are unreliable as the scheduling mechanism for round robin check execution, and etcd will not produce precise round robin behavior.

When you enable round robin scheduling on PostgreSQL, any existing round robin scheduling will stop

and migrate to PostgreSQL as entities check in with keepalives. Sensu will gradually delete the existing etcd scheduler state as keepalives on the etcd scheduler keys expire over time.

Interval scheduling

You can schedule a check to be executed at regular intervals using the `interval` and `publish` check attributes. For example, to schedule a check to execute every 60 seconds, set the `interval` attribute to `60` and the `publish` attribute to `true`.

> ***NOTE**: When creating an interval check, Sensu calculates an initial offset to splay the check's first scheduled request. This helps balance the load of both the backend and the agent and may result in a delay before initial check execution.*

Example interval check

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: interval_check
spec:
  command: check-cpu.sh -w 75 -c 90
  handlers:
  - slack
  interval: 60
  publish: true
  subscriptions:
  - system
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "interval_check"
  },
  "spec": {
```

```
    "command": "check-cpu.sh -w 75 -c 90",
    "subscriptions": ["system"],
    "handlers": ["slack"],
    "interval": 60,
    "publish": true
  }
}
```

Cron scheduling

You can also schedule checks using cron syntax.

Examples of valid cron values include:

- `cron: CRON_TZ=Asia/Tokyo * * * * *`
- `cron: TZ=Asia/Tokyo * * * * *`
- `cron: '* * * * *'`

> ***NOTE**: If you're using YAML to create a check that uses cron scheduling and the first character of the cron schedule is an asterisk (`*`), place the entire cron schedule inside single or double quotes (for example, `cron: '* * * * *'`).*

Example cron checks

To schedule a check to execute once a minute at the start of the minute, set the `cron` attribute to `* * * * *` and the `publish` attribute to `true`:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: cron_check
spec:
  command: check-cpu.sh -w 75 -c 90
  cron: '* * * * *'
  handlers:
```

```
  - slack
  publish: true
  subscriptions:
  - system
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "cron_check"
  },
  "spec": {
    "command": "check-cpu.sh -w 75 -c 90",
    "subscriptions": ["system"],
    "handlers": ["slack"],
    "cron": "* * * * *",
    "publish": true
  }
}
```

Use a prefix of `TZ=` or `CRON_TZ=` to set a timezone for the `cron` attribute:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: cron_check
spec:
  check_hooks: null
  command: hi
  cron: CRON_TZ=Asia/Tokyo * * * * *
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 0
  low_flap_threshold: 0
```

```
  output_metric_format: ""
  output_metric_handlers: null
  output_metric_tags: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets: null
  stdin: false
  subdue: null
  subscriptions:
  - sys
  timeout: 0
  ttl: 0
```

JSON

```
{
   "type": "CheckConfig",
   "api_version": "core/v2",
   "metadata": {
      "name": "cron_check"
   },
   "spec": {
      "check_hooks": null,
      "command": "hi",
      "cron": "CRON_TZ=Asia/Tokyo * * * * *",
      "env_vars": null,
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 0,
      "low_flap_threshold": 0,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "output_metric_tags": null,
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": null,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
         "sys"
```

```
        ],
        "timeout": 0,
        "ttl": 0
    }
}
```

Ad hoc scheduling

In addition to automatic execution, you can create checks to be scheduled manually using core/v2/checks API endpoints. To create a check with ad-hoc scheduling, set the `publish` attribute to `false` in addition to an `interval` or `cron` schedule.

Example ad hoc check

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: ad_hoc_check
spec:
  command: check-cpu.sh -w 75 -c 90
  handlers:
  - slack
  interval: 60
  publish: false
  subscriptions:
  - system
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "ad_hoc_check"
  },
  "spec": {
    "command": "check-cpu.sh -w 75 -c 90",
```

```
    "subscriptions": ["system"],
    "handlers": ["slack"],
    "interval": 60,
    "publish": false
  }
}
```

Proxy checks

Sensu supports running proxy checks that associate events with a entity that isn't actually executing the check, regardless of whether that entity is an agent entity or a proxy entity. Proxy entities allow Sensu to monitor external resources on systems and devices where a Sensu agent cannot be installed, like a network switch or a website. You can create a proxy check using `proxy_entity_name` or `proxy_requests`.

When you create a proxy check, make sure the check definition includes a subscription that matches the subscription of at least one agent entity to define which agents will run the check. Proxy entities do not use subscriptions.

To avoid duplicate events, use the `round_robin` check attribute with proxy checks. Read Round robin checks and Proxy entities and round robin scheduling to learn more.

Read Monitor external resources with proxy entities to learn how to create proxy checks to generate events for one or more proxy entities.

Use a proxy check to monitor a proxy entity

When executing checks that include a `proxy_entity_name`, Sensu agents report the resulting events under the specified proxy entity instead of the agent entity. If the proxy entity doesn't exist, Sensu creates the proxy entity when the backend receives the event.

Example proxy check using `proxy_entity_name`

The following proxy check runs every 60 seconds, cycling through the agents with the `run_proxies` subscription alphabetically according to the agent name, for the proxy entity `sensu-site`.

YML

```
```

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: proxy_check
spec:
  command: http_check.sh https://sensu.io
  handlers:
  - slack
  interval: 60
  proxy_entity_name: sensu-site
  publish: true
  round_robin: true
  subscriptions:
  - run_proxies
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "proxy_check"
  },
  "spec": {
    "command": "http_check.sh https://sensu.io",
    "subscriptions": ["run_proxies"],
    "handlers": ["slack"],
    "interval": 60,
    "publish": true,
    "round_robin": true,
    "proxy_entity_name": "sensu-site"
  }
}
```

Use a proxy check to monitor multiple proxy entities

The `proxy_requests` check attributes allow Sensu to run a check for each entity that matches the expressions specified in the `entity_attributes`, resulting in observability events that represent each matching proxy entity. The entity attributes must match exactly as stated.

No variables or directives have any special meaning, but you can use Sensu query expressions to perform more complicated filtering on the available value, such as finding entities with a particular class or label.

Combine `proxy_requests` attributes with with token substitution as shown in the example proxy check below to monitor multiple entities using a single check definition.

Example proxy check using `proxy_requests`

The following proxy check runs every 60 seconds, cycling through the agents with the `run_proxies` subscription alphabetically according to the agent name, for all existing proxy entities with the custom label `proxy_type` set to `website`.

This check uses token substitution to import the value of the custom entity label `url` to complete the check command.Read the entities reference for information about adding custom labels to entities.

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: proxy_check_proxy_requests
spec:
  command: http_check.sh {{ .labels.url }}
  handlers:
  - slack
  interval: 60
  proxy_requests:
    entity_attributes:
    - entity.labels.proxy_type == 'website'
  publish: true
  round_robin: true
  subscriptions:
  - run_proxies
```

JSON

```
{
  "type": "CheckConfig",
```

```
  "api_version": "core/v2",
  "metadata": {
    "name": "proxy_check_proxy_requests"
  },
  "spec": {
    "command": "http_check.sh {{ .labels.url }}",
    "subscriptions": ["run_proxies"],
    "handlers": ["slack"],
    "interval": 60,
    "publish": true,
    "proxy_requests": {
      "entity_attributes": [
        "entity.labels.proxy_type == 'website'"
      ]
    },
    "round_robin": true
  }
}
```

Fine-tune proxy check scheduling with splay

Use the `splay` and `splay_coverage` attributes to distribute proxy check executions across the check interval.

To continue the example proxy_check_proxy_requests check, if the check matches three proxy entities, you will get a single burst of three check executions (with the resulting events) every 60 seconds. Use the `splay` and `splay_coverage` attributes to distribute the three check executions over the specified check interval instead of all at the same time.

The following example adds `splay` set to `true` and `splay_coverage` set to `90` within the `proxy_requests` object. With this addition, instead of three check executions in a single burst every 60 seconds, Sensu will distribute the three check executions evenly across a 54-second period (90% of the 60-second interval):

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: proxy_check_proxy_requests
```

```
spec:
  command: http_check.sh {{ .labels.url }}
  handlers:
  - slack
  interval: 60
  proxy_requests:
    entity_attributes:
    - entity.labels.proxy_type == 'website'
    splay: true
    splay_coverage: 90
  publish: true
  round_robin: true
  subscriptions:
  - run_proxies
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "proxy_check_proxy_requests"
  },
  "spec": {
    "command": "http_check.sh {{ .labels.url }}",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "proxy_requests": {
      "entity_attributes": [
        "entity.labels.proxy_type == 'website'"
      ],
      "splay": true,
      "splay_coverage": 90
    },
    "publish": true,
    "round_robin": true,
    "subscriptions": [
      "run_proxies"
    ]
  }
```

```
}
```

Check token substitution

Sensu check definitions may include attributes that you wish to override on an entity-by-entity basis. For example, check commands, which may include command line arguments for controlling the behavior of the check command, may benefit from entity-specific thresholds. Sensu check tokens are check definition placeholders that the Sensu agent will replace with the corresponding entity definition attribute values (including custom attributes).

Learn how to use check tokens with the Sensu tokens reference documentation.

> ***NOTE**: Check tokens are processed before check execution, so token substitutions will not apply to check data delivered via the local agent socket input.*

Subdues

Use the `subdues` attribute in check definitions to set specific periods of time when Sensu should not execute the check. Subdues allow you to schedule alert-free periods of time, such as during sleeping hours, weekends, or special maintenance periods.

You can set more than one subdue at a time. Each subdue includes a begin and end time as well as how often to repeat the subdue, if desired.

For example, this check will be subdued (in other words, will not be executed) from 5:00 p.m. until 8:00 a.m. PDT on every weekday, and for the entire day on weekends, as well as every Friday from 10:00 until 11:00 a.m. PDT:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
spec:
  command: check-cpu-usage -w 75 -c 90
```

```
  interval: 60
  handlers:
  - slack
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  subdues:
  - begin: "2022-04-18T17:00:00-07:00"
    end: "2022-04-19T08:00:00-07:00"
    repeat:
    - weekdays
  - begin: "2022-04-23T00:00:00-07:00"
    end: "2022-04-23T23:59:59-07:00"
    repeat:
    - weekends
  - begin: "2022-04-22T10:00:00-07:00"
    end: "2022-04-22T11:00:00-07:00"
    repeat:
    - fridays
  subscriptions:
  - system
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu"
  },
  "spec": {
    "command": "check-cpu-usage -w 75 -c 90",
    "interval": 60,
    "handlers": [
      "slack"
    ],
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
```

```
    "subdues": [
      {
        "begin": "2022-04-18T17:00:00-07:00",
        "end": "2022-04-19T08:00:00-07:00",
        "repeat": [
          "weekdays"
        ]
      },
      {
        "begin": "2022-04-23T00:00:00-07:00",
        "end": "2022-04-23T23:59:59-07:00",
        "repeat": [
          "weekends"
        ]
      },
      {
        "begin": "2022-04-22T10:00:00-07:00",
        "end": "2022-04-22T11:00:00-07:00",
        "repeat": [
          "fridays"
        ]
      }
    ],
    "subscriptions": [
      "system"
    ]
  }
}
```

Subdues and repeat

If you include a `repeat` array in a `subdues` object, Sensu will start the subdue period on the date you specify. After the first subdue, Sensu uses the `begin` and `end` values only to determine the *time* of day to start and stop the subdue.

***NOTE**: Check subdue repeats are based on the specified `begin` and `end` times and not duration or the difference between the `begin` and `end` times. Read Repeat and multi-day subdues for more information.*

In the above example, on April 18, 2022, Sensu will apply the `weekdays` subdue at 5:00 p.m. PDT and end it on April 19 at 8:00 a.m. PDT. On April 19, Sensu will apply the `weekdays` subdue again at 5:00 p.m. PDT and end it on April 20 at 8:00 a.m. PDT, and so on.

Valid values for repeat arrays

This table lists and describes valid values for the `repeat` array:

Value	Description
`mondays` `tuesdays` `wednesdays` `thursdays` `fridays` `saturdays` `sundays`	Subdue on the specified day, at the same time of day
`weekdays`	Subdue on all Mondays, Tuesdays, Wednesdays, Thursdays, and Fridays, at the same time of day
`weekends`	Subdue on all Saturdays and Sundays, at the same time of day
`daily`	Subdue once every day, at the same time of day
`weekly`	Subdue once per week on the same day, at the same time of day
`monthly`	Subdue once per month on the same day, at the same time of day
`annually`	Subdue once per year on the same day, at the same time of day

Repeat and multi-day subdues

Because repeat schedules for subdues are based only on the specified time of day, you may need to configure more than one repeat for multi-day subdues.

For example, suppose that you want to subdue a check on the weekends. You might set a repeat that starts on a Friday at 5:00 p.m. PDT and ends on Monday at 8:00 a.m. PDT:

YML

```
```

```yaml
subdues:
- begin: "2022-05-06T17:00:00-07:00"
  end: "2022-05-09T08:00:00-07:00"
  repeat:
  - fridays
```

JSON

```
{
  "subdues": [
    {
      "begin": "2022-05-06T17:00:00-07:00",
      "end": "2022-05-09T08:00:00-07:00",
      "repeat": [
        "fridays"
      ]
    }
  ]
}
```

The first weekend, your repeat will work as expected to subdue the check from 5:00 p.m. PDT on Friday until 8:00 a.m. PDT on Monday.

After the first weekend, the subdue will start as expected at 5:00 p.m. PDT on Friday, but it will expire at 8:00 a.m. PDT on *Saturday* instead of Monday. This is because after the first instance, repeats are based only on the specified `begin` and `end` times. Sensu uses the dates to schedule only the first subdue.

Instead, use the following three-part configuration to achieve the desired repeat schedule (every Friday at 5:00 p.m. PDT until Monday at 8:00 a.m. PDT):

YML

```yaml
subdues:
- begin: "2022-05-06T17:00:00-07:00"
  end: "2022-05-06T23:59:59-07:00"
  repeat:
  - fridays
- begin: "2022-05-07T00:00:00-07:00"
  end: "2022-05-07T23:59:59-07:00"
```

```
  repeat:
  - weekends
- begin: "2022-05-09T00:00:00-07:00"
  end: "2022-05-09T08:00:00-07:00"
  repeat:
  - mondays
```

JSON

```
{
  "subdues": [
    {
      "begin": "2022-05-06T17:00:00-07:00",
      "end": "2022-05-06T23:59:59-07:00",
      "repeat": [
        "fridays"
      ]
    },
    {
      "begin": "2022-05-07T00:00:00-07:00",
      "end": "2022-05-07T23:59:59-07:00",
      "repeat": [
        "weekends"
      ]
    },
    {
      "begin": "2022-05-09T00:00:00-07:00",
      "end": "2022-05-09T08:00:00-07:00",
      "repeat": [
        "mondays"
      ]
    }
  ]
}
```

With this configuration, the repeat schedule will subdue the check every Friday from 5:00 p.m. PDT until midnight, the entire 24 hours on every Saturday and Sunday, and every Monday from midnight until 8:00 a.m. PDT.

Check hooks

Check hooks are commands run by the Sensu agent in response to the result of check command execution. The Sensu agent will execute the appropriate configured hook command, depending on the check execution status (for example, `0`, `1`, or `2`).

Learn how to use check hooks with the Sensu hooks reference documentation.

Check specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For checks in this version of Sensu, this attribute should always be `core/v2`.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata	
description	Top-level collection of metadata about the check, including `name`,

	`namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the check definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: sensu-site-perf` `  namespace: development` `  created_by: admin` `  labels:` `  region: us-west-1` `  environment: dev` `  annotations:` `  slack-channel: "#monitoring"` `  managed-by: ops` `  playbooks: www.playbooks-example.url` **JSON** `{` `  "metadata": {` `  "name": "sensu-site-perf",` `  "namespace": "development",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1",` `  "environment": "dev"` `  },` `  "annotations": {` `  "slack-channel": "#monitoring",` `  "managed-by": "ops",` `  "playbooks": "www.playbooks-example.url"` `  }` `  }` `}`

spec	
description	Top-level map that includes the check spec attributes.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	YML

```yaml
spec:
  check_hooks: null
  command: collect.sh
  discard_output: true
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: prometheus_text
  output_metric_tags:
  - name: instance
    value: '{{ .name }}'
  - name: namespace
    value: "{{ .namespace }}"
  - name: service
    value: '{{ .labels.service }}'
  output_metric_thresholds:
    - name: system_mem_used
      tags: null
      null_status: 0
      thresholds:
      - max: "75.0"
        min: ""
        status: 1
      - max: "90.0"
        min: ""
        status: 2
```

```
    - name: system_host_processes
      tags:
      - name: namespace
        value: production
      null_status: 0
      thresholds:
      - max: "50"
        min: "5"
        status: 1
      - max: "75"
        min: "2"
        status: 2
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: prometheus_gateway_workflows
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets: null
  stdin: false
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "spec": {
    "check_hooks": null,
    "command": "collect.sh",
    "discard_output": true,
    "env_vars": null,
    "handlers": [

    ],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "prometheus_text",
    "output_metric_tags": [
```

```
        {
          "name": "instance",
          "value": "{{ .name }}"
        },
        {
          "name": "namespace",
          "value": "{{ .namespace }}"
        },
        {
          "name": "service",
          "value": "{{ .labels.service }}"
        }
      ],
      "output_metric_thresholds": [
        {
          "name": "system_mem_used",
          "tags": null,
          "null_status": 0,
          "thresholds": [
            {
              "max": "75.0",
              "min": "",
              "status": 1
            },
            {
              "max": "90.0",
              "min": "",
              "status": 2
            }
          ]
        },
        {
          "name": "system_host_processes",
          "tags": [
            {
              "name": "namespace",
              "value": "production"
            }
          ],
          "null_status": 0,
          "thresholds": [
            {
```

```
              "max": "50",
              "min": "5",
              "status": 1
            },
            {
              "max": "75",
              "min": "2",
              "status": 2
            }
          ]
        }
      ],
      "pipelines": [
        {
          "type": "Pipeline",
          "api_version": "core/v2",
          "name": "prometheus_gateway_workflows"
        }
      ],
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": null,
      "stdin": false,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "ttl": 0
    }
}
```


type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Checks should always be type `CheckConfig`.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.

type	String
example	**YML** `type: CheckConfig` **JSON** `{ "type": "CheckConfig" }`

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations: slack-channel: "#monitoring" managed-by: ops playbooks: www.playbooks-example.url` **JSON**

```
{
  "annotations": {
    "slack-channel": "#monitoring",
    "managed-by": "ops",
    "playbooks": "www.playbooks-example.url"
  }
}
```

created_by

description	Username of the Sensu user who created the check or last updated the check. Sensu automatically populates the `created_by` field when the check is created or updated.
required	false
type	String

YML

example

```
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.

	Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example

YML

```
labels:
  region: us-west-1
  environment: dev
```

JSON

```
{
  "labels": {
    "region": "us-west-1",
    "environment": "dev"
  }
}
```

name

description	Unique string used to identify the check. Check names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each check must have a unique name within its namespace.
required	true
type	String

example

YML

```
name: sensu-site-perf
```

JSON

```
{
  "name": "sensu-site-perf"
}
```

namespace	
description	Sensu RBAC namespace that the check belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: development` **JSON** `{ "namespace": "development" }`

Spec attributes

***NOTE**: Spec attributes are not required when sending an HTTP `POST` request to the agent events API or the backend core/v2/events API. When doing so, the spec attributes are listed as individual top-level attributes in the check definition instead.*

check_hooks	
description	Array of check response types with respective arrays of Sensu hook names. Sensu hooks are commands run by the Sensu agent in response to the result of the check command execution. Hooks are executed in order of precedence based on their severity type: `1` to `255`, `ok`, `warning`, `critical`, `unknown`, and finally `non-zero`.
required	false
type	Array
example	**YML** `check_hooks:` `- '1':` `  - playbook-warning` `  - collect-diagnostics` `- critical:` `  - playbook-critical` `  - collect-diagnostics` `  - process-tree` **JSON** `{` `  "check_hooks": [` `  {` `  "1": [` `  "playbook-warning",` `  "collect-diagnostics"` `  ]` `  },` `  {` `  "critical": [` `  "playbook-critical",` `  "collect-diagnostics",` `  "process-tree"` `  ]` `  }` `  ]` `}`

command

description	Check command to be executed.
required	true
type	String
example	**YML** `command: http-perf --url https://sensu.io --warning 1s --critical 2s` **JSON** `{ "command": "http-perf --url https://sensu.io --warning 1s --critical 2s" }`

cron

description	When the check should be executed, using cron syntax or a predefined schedule. Use a prefix of `TZ=` or `CRON_TZ=` to set a timezone for the cron attribute. ***NOTE**: If you're using YAML to create a check that uses cron scheduling and the first character of the cron schedule is an asterisk (`*`), place the entire cron schedule inside single or double quotes (for example, `cron: '* * * * *'`).*
required	true (unless `interval` is configured)
type	String

YML

example

```
cron: 0 0 * * *
```

JSON

```
{
  "cron": "0 0 * * *"
}
```

env_vars

description

Array of environment variables to use with command execution.

> ***NOTE**: To add `env_vars` to a check, use `sensuctl create`.*

required

false

type

Array

YML

example

```
env_vars:
- APP_VERSION=2.5.0
- CHECK_HOST=my.host.internal
```

JSON

```
{
  "env_vars": [
    "APP_VERSION=2.5.0",
    "CHECK_HOST=my.host.internal"
  ]
}
```

handlers

description	Array of Sensu event handlers (names) to use for events created by the check. Each array item must be a string. ***NOTE**: The names of Sumo Logic metrics handlers and TCP stream handlers are not valid values for the handlers array. Only traditional handlers are valid for the handlers array.* *To use Sumo Logic metrics or TCP stream handlers, include them in a pipeline workflow and reference the pipeline name in the check pipelines array.*
required	false
type	Array

YML

```
handlers:
- pagerduty
- slack
```

JSON

```
{
  "handlers": [
    "pagerduty",
    "slack"
  ]
}
```

high_flap_threshold

description	Flap detection high threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios. Read the event reference to learn more about how Sensu uses the

	`high_flap_threshold` value.
required	true (if `low_flap_threshold` is configured)
type	Integer
example	**YML** `high_flap_threshold: 60` **JSON** `{ "high_flap_threshold": 60 }`

interval

description	How often the check is executed. In seconds.
required	true (unless `cron` is configured)
type	Integer
example	**YML** `interval: 60` **JSON** `{ "interval": 60 }`

low_flap_threshold

description	Flap detection low threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios. Read the event reference to learn more about how Sensu uses the `low_flap_threshold` value.
required	false
type	Integer

YML

example

```
low_flap_threshold: 20
```

JSON

```
{
  "low_flap_threshold": 20
}
```

output_metric_format

description	Metric format generated by the check command. Sensu supports the following metric formats: `nagios_perfdata` (Nagios Performance Data) `graphite_plaintext` (Graphite Plaintext Protocol) `influxdb_line` (InfluxDB Line Protocol) `opentsdb_line` (OpenTSDB Data Specification) `prometheus_text` (Prometheus Exposition Text) When a check includes an `output_metric_format`, Sensu will extract the metrics from the check output and add them to the event data in Sensu metric format. Read Collect metrics with Sensu checks.
required	false
type	String

YML

example

```
output_metric_format:
- nagios_perfdata
```

JSON

```
{
  "output_metric_format": [
    "nagios_perfdata"
  ]
}
```

output_metric_handlers

description	Array of Sensu handlers to use for events created by the check. Each array item must be a string. Use `output_metric_handlers` in place of the `handlers` attribute if `output_metric_format` is configured. Metric handlers must be able to process Sensu metric format. The Sensu InfluxDB handler provides an example.
required	false
type	Array
example	**YML** `output_metric_handlers:` `- influx-db`

JSON

```
{
  "output_metric_handlers": [
    "influx-db"
  ]
}
```

output_metric_tags

description	Custom tags to enrich metric points produced by check output metric

extraction. One name/value pair make up a single tag. The `output_metric_tags` array can contain multiple tags.

You can use check token substitution for the `value` attribute in output metric tags.

required	false
type	Array

YML

example

```
output_metric_tags:
- name: instance
  value: "{{ .name }}"
- name: region
  value: "{{ .labels.region }}"
```

JSON

```
{
  "output_metric_tags": [
    {
      "name": "instance",
      "value": "{{ .name }}"
    },
    {
      "name": "region",
      "value": "{{ .labels.region }}"
    }
  ]
}
```

output_metric_thresholds

description

Array of metric names and threshold values to compare to check output metrics for metric threshold evaluation.

***NOTE**: To apply metric threshold evaluation, check definitions*

must include the `output_metric_format` attribute with a value that specifies one of Sensu's supported output metric formats.

required	false
type	Array

YML

example

```
output_metric_thresholds:
- name: system_mem_used
  tags: ''
  null_status: 0
  thresholds:
  - max: '75.0'
    min: ''
    status: 1
  - max: '90.0'
    min: ''
    status: 2
- name: system_host_processes
  tags:
  - name: namespace
    value: production
  null_status: 0
  thresholds:
  - max: '50'
    min: '5'
    status: 1
  - max: '75'
    min: '2'
    status: 2
```

JSON

```
{
  "output_metric_thresholds": [
    {
      "name": "system_mem_used",
      "tags": null,
      "null_status": 0,
```

```
      "thresholds": [
        {
          "max": "75.0",
          "min": "",
          "status": 1
        },
        {
          "max": "90.0",
          "min": "",
          "status": 2
        }
      ]
    },
    {
      "name": "system_host_processes",
      "tags": [
        {
          "name": "namespace",
          "value": "production"
        }
      ],
      "null_status": 0,
      "thresholds": [
        {
          "max": "50",
          "min": "5",
          "status": 1
        },
        {
          "max": "75",
          "min": "2",
          "status": 2
        }
      ]
    }
  ]
}
```

pipelines

description	Name, type, and API version for the pipelines to use for event processing. All the observability events that the check produces will be processed according to the pipelines listed in the pipeline array. Read pipelines attributes for more information.
required	false
type	Array
example	**YML** `pipelines:` `- type: Pipeline` `  api_version: core/v2` `  name: incident_alerts` **JSON** `{` `  "pipelines": [` `  {` `  "type": "Pipeline",` `  "api_version": "core/v2",` `  "name": "incident_alerts"` `  }` `  ]` `}`

proxy_entity_name

description	Entity name. Used to create a proxy entity for an external resource (for example, a network switch).
required	false
type	String
validated	`\A[\w\.\-]+\z`

example	**YML** `proxy_entity_name: switch-dc-01` **JSON** `{ "proxy_entity_name": "switch-dc-01" }`

proxy_requests

description	Assigns a check to run for multiple entities according to their `entity_attributes`. In the example below, the check executes for all entities with entity class `proxy` and the custom proxy type label `website`. The `proxy_requests` attributes allow you to reuse check definitions for a group of entities. For more information, read Proxy requests attributes and Monitor external resources with proxy entities.
required	false
type	Hash
example	**YML** `proxy_requests: entity_attributes: - entity.entity_class == 'proxy' - entity.labels.proxy_type == 'website' splay: true splay_coverage: 90` **JSON** `{ "proxy_requests": { "entity_attributes": [ "entity.entity_class == 'proxy'", "entity.labels.proxy_type == 'website'"`

```
    ],
    "splay": true,
    "splay_coverage": 90
  }
}
```

publish

description	`true` if check requests are published for the check. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	**YML** `publish: true` **JSON** `{ "publish": true }`

round_robin

description	When set to `true`, Sensu executes the check once per interval, cycling through each subscribing agent in turn. Read round robin checks for more information. Use the `round_robin` attribute with proxy checks to avoid duplicate events and distribute proxy check executions evenly across multiple agents. Read about proxy checks for more information.

	To use check `ttl` and `round_robin` together, your check configuration must also specify a `proxy_entity_name`. If you do not specify a `proxy_entity_name` when using check `ttl` and `round_robin` together, your check will stop executing.
required	false
type	Boolean
default	`false`
example	**YML** `round_robin: true` **JSON** `{ "round_robin": true }`

runtime_assets

description	Array of Sensu dynamic runtime assets (names). Required at runtime for the execution of the `command`.
required	false
type	Array
example	**YML** `runtime_assets:` `- http-checks` **JSON** `{ "runtime_assets": [ "http-checks"`

```
  ]
}
```

scheduler

description	Type of scheduler that schedules the check. Sensu automatically sets the `scheduler` value and overrides any user-entered values. Value may be: ¬ `memory` for checks scheduled in-memory ¬ `etcd` for checks scheduled with etcd leases and watchers (check attribute `round_robin: true` and etcd used for event storage) ¬ `postgres` for checks scheduled with PostgreSQL using transactions and asynchronous notification (check attribute `round_robin: true` and PostgreSQL used for event storage with datastore attribute `enable_round_robin: true`)
required	false
type	String

YML

```
scheduler: postgres
```

JSON

```
{
  "scheduler": "postgres"
}
```

secrets

description	Array of the name/secret pairs to use with command execution.
required	false
type	Array
example	**YML** `secrets:` `- name: PAGERDUTY_TOKEN` `  secret: sensu-pagerduty-token` **JSON** `{` `  "secrets": [` `  {` `  "name": "PAGERDUTY_TOKEN",` `  "secret": "sensu-pagerduty-token"` `  }` `  ]` `}`

silenced

description	Silences that apply to the check.
type	Array
example	**YML** `silenced:` `- "*:routers"` **JSON** `{` `  "silenced": [` `  "*:routers"` `  ]`

```
}
```

stdin

description	`true` if the Sensu agent writes JSON serialized Sensu entity and check data to the command process' stdin. The command must expect the JSON data via stdin, read it, and close stdin. Otherwise, `false`. This attribute cannot be used with existing Sensu check plugins or Nagios plugins because the Sensu agent will wait indefinitely for the check process to read and close stdin.
required	false
type	Boolean
default	`false`
example	**YML** `stdin: true` **JSON** `{ "stdin": true }`

subdue (placeholder)

description	Use the `subdues` attribute to stop check execution during specific periods. This `subdue` attribute appears in check definitions by default, but it is a placeholder and should not be modified.
example	**YML** `subdue: null`

JSON

```
{
  "subdue": null
}
```

subdues	
description	Specific periods of time when Sensu should not send alerts based on the events the check produces. Use to schedule alert-free periods of time, such as during sleeping hours, weekends, or special maintenance periods. Read subdues attributes for more information.
required	false
type	Array

YML

example

```
subdues:
  - begin: "2022-04-18T17:00:00-07:00"
    end: "2022-04-19T08:00:00-07:00"
    repeat:
    - weekdays
  - begin: "2022-04-23T00:00:00-07:00"
    end: "2022-04-23T23:59:59-07:00"
    repeat:
    - weekends
  - begin: "2022-04-22T10:00:00-07:00"
    end: "2022-04-22T11:00:00-07:00"
    repeat:
    - fridays
```

JSON

```
{
  "subdues": [
    {
      "begin": "2022-04-18T17:00:00-07:00",
```

```
      "end": "2022-04-19T08:00:00-07:00",
      "repeat": [
        "weekdays"
      ]
    },
    {
      "begin": "2022-04-23T00:00:00-07:00",
      "end": "2022-04-23T23:59:59-07:00",
      "repeat": [
        "weekends"
      ]
    },
    {
      "begin": "2022-04-22T10:00:00-07:00",
      "end": "2022-04-22T11:00:00-07:00",
      "repeat": [
        "fridays"
      ]
    }
  ]
}
```

subscriptions	
description	Array of Sensu entity subscriptions that check requests will be sent to. The array cannot be empty and its items must each be a string.
required	true
type	Array

YML

example

```
subscriptions:
- system
```

JSON

```
{
```

```
  "subscriptions": [
    "system"
  ]
}
```

timeout

description	Check execution duration timeout (hard stop). In seconds.
required	false
type	Integer
example	**YML** `timeout: 30` **JSON** `{ "timeout": 30 }`

ttl

description	The time-to-live (TTL) until check results are considered stale. In seconds. If an agent stops publishing results for the check and the TTL expires, an event will be created for the agent's entity. The check `ttl` must be greater than the check `interval` and should allow enough time for the check execution and result processing to complete. For example, for a check that has an `interval` of `60` (seconds) and a `timeout` of `30` (seconds), the appropriate `ttl` is at least `90` (seconds). To use check `ttl` and `round_robin` together, your check

configuration must also specify a `proxy_entity_name`. If you do not specify a `proxy_entity_name` when using check `ttl` and `round_robin` together, your check will stop executing.

> ***NOTE**: Adding TTLs to checks adds overhead, so use the `ttl` attribute sparingly.*

required	false
type	Integer
example	**YML** `ttl: 100` **JSON** `{ "ttl": 100 }`

Check output truncation attributes

max_output_size	
description	Maximum size of stored check outputs. In bytes. When set to a non-zero value, the Sensu backend truncates check outputs larger than this value before storing to etcd. `max_output_size` does not affect data sent to Sensu filters, mutators, and handlers. As of Sensu Go 6.8.2, when check output is truncated due to the `max_output_size` configuration, the events the check produces will include a `sensu.io/output_truncated_bytes` label.
required	false
type	Integer

example	**YML** `max_output_size: 1024` **JSON** `{ "max_output_size": 1024 }`

discard_output	
description	If `true`, discard check output after extracting metrics. No check output will be sent to the Sensu backend. Otherwise, `false`.
required	false
type	Boolean
example	**YML** `discard_output: true` **JSON** `{ "discard_output": true }`

`output_metric_tags` *attributes*

name	
description	Name for the output metric tag.

required	true
type	String
example	**YML** `name: instance` **JSON** `{ "name": "instance" }`

value

description	Value for the output metric tag. Use check token substitution syntax for the `value` attribute, with dot-notation access to any event attribute.
required	true
type	String
example	**YML** `value: {{ .name }}` **JSON** `{ "value": "{{ .name }}" }`

`output_metric_thresholds` *attributes*

name

description	Name of the metric to use for metric threshold evaluation. Must match the event.metrics.points[].name value for a metric point in the check results. ***NOTE**: To produce values for the output metrics you specify, the check definition must include a valid `output_metric_format`.*
required	true
type	String
example	**YML** ``` name: system_host_processes ``` **JSON** ``` { "name": "system_host_processes" } ```

null_status

description	Event check status to use if a metric specified for metric threshold evaluation is missing from the event data. ***NOTE**: Sensu only overrides the event check status if it is less than the specified `null_status` value.*
required	false
type	Integer
default	`0`

example	**YML** ```yaml null_status: 0 ``` **JSON** ```json { "null_status": 0 } ```

tags

description	Tags of the metric to use for metric threshold evaluation. If provided, must match the event.metrics.points[].tags name and value for a metric point in the check results. Read tags attributes for more information.
required	false
type	Array
example	**YML** ```yaml tags: - name: namespace value: production ``` **JSON** ```json { "tags": [{ "name": "namespace", "value": "production" }] } ```

thresholds	
description	Rules to apply for metric threshold evaluation. Read thresholds attributes for more information.
required	true
type	Array

YML

example

```yaml
thresholds:
- max: '50'
  min: '5'
  status: 1
- max: '75'
  min: '2'
  status: 2
```

JSON

```json
{
  "thresholds": [
    {
      "max": "50",
      "min": "5",
      "status": 1
    },
    {
      "max": "75",
      "min": "2",
      "status": 2
    }
  ]
}
```

Pipelines attributes

api_version

description	The Sensu API group and version for the pipeline. For pipelines in this version of Sensu, the api_version should always be `core/v2`.
required	true
type	String
default	`null`
example	**YML** ```yaml api_version: core/v2 ``` **JSON** ```json { "api_version": "core/v2" } ```

name

description	Name of the Sensu pipeline for the check to use.
required	true
type	String
default	`null`
example	**YML** ```yaml name: incident_alerts ``` **JSON** ```json { "name": "incident_alerts" ```

```
}
```

type	
description	The `sensuctl create` resource type for the pipeline. Pipelines should always be type `Pipeline`.
required	true
type	String
default	`null`
example	**YML** `type: Pipeline` **JSON** `{ "type": "Pipeline" }`

Proxy requests attributes

entity_attributes	
description	Sensu entity attributes to match entities in the registry using Sensu query expressions.
required	false
type	Array
example	**YML** `entity_attributes:`

```
- entity.entity_class == 'proxy'
- entity.labels.proxy_type == 'website'
```

JSON

```
{
  "entity_attributes": [
    "entity.entity_class == 'proxy'",
    "entity.labels.proxy_type == 'website'"
  ]
}
```

splay

description	`true` if proxy check requests should be splayed, published evenly over a window of time, determined by the check interval and a configurable `splay_coverage` percentage. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	**YML** `splay: true` **JSON** `{ "splay": true }`

splay_coverage

description	**Percentage** of the check interval over which Sensu can execute the check for all applicable entities, as defined in the entity attributes. Sensu uses the splay_coverage attribute to determine the period of time to publish check requests over, before the next check interval begins. For example, if a check's interval is 60 seconds and `splay_coverage` is 90, Sensu will distribute its proxy check requests evenly over a time window of 54 seconds (60 seconds * 90%). This leaves 6 seconds after the last proxy check execution before the the next round of proxy check requests for the same check.
required	`true` if `splay` attribute is set to `true` (otherwise, `false`)
type	Integer
example	**YML** `splay_coverage: 90` **JSON** `{ "splay_coverage": 90 }`

`secrets` *attributes*

name	
description	Name of the secret defined in the executable command. Becomes the environment variable presented to the check. Read Use secrets management in Sensu for more information.
required	true
type	String
example	**YML** `name: ANSIBLE_HOST`

JSON

```
{
  "name": "ANSIBLE_HOST"
}
```

secret	
description	Name of the Sensu secret resource that defines how to retrieve the secret.
required	true
type	String
example	**YML** `secret: sensu-ansible-host` **JSON** `{ "secret": "sensu-ansible-host" }`

Tags attributes

name	
description	Tag name for the metric to use for metric threshold evaluation. If provided, must match the event.metrics.points[].tags.name value for a metric point in the check results.

***NOTE**: If provided, you must also provide the value for the same metric point tag.*

required	false
type	String

YML

example

```
name: namespace
```

JSON

```
{
  "name": "namespace"
}
```

value

description

Tag value of the metric to use for metric threshold evaluation. If provided, must match the event.metrics.points[].tags.value value for a metric point in the check results.

***NOTE**: If provided, you must also provide the name for the same metric point tag.*

required	false
type	String

YML

example

```
value: production
```

JSON

```
{
```

```
  "value": "production"
}
```

Thresholds attributes

max	
description	Maximum threshold for the metric for metric threshold evaluation. You must provide a thresholds `max` value if you do not provide a `min` value.
required	false (if a thresholds `min` value is provided)
type	String

YML

```
max: '75'
```

JSON

```
{
  "max": "75"
}
```

min	
description	Minimum threshold for the metric for metric threshold evaluation. You must provide a thresholds `min` value if you do not provide a `max` value.
required	false (if a thresholds `max` value is provided)
type	String

YML

example

```
min: '2'
```

JSON

```
{
  "min": "2"
}
```

status

description

Event check status to use if the check's output metric value is equal to or greater than the specified `max` threshold or equal to or less than the specified `min` threshold in metric threshold evaluation.

> ***NOTE**: Sensu only overrides the event check status if it is less than the specified threshold `status` value.*

You can specify any status value, but event annotations based on threshold status will display `unknown` if the status does not equal `0`, `1`, or `2`.

required

true

type

Integer

YML

example

```
status: 2
```

JSON

```
{
  "status": 2
}
```

`subdues` *attributes*

begin

description	Date and time at which the subdue should begin. In RFC 3339 format with numeric zone offset (`2022-01-01T07:30:00-07:00` or `2022-01-01T14:30:00Z`).
required	true
type	String

YML

example

```yaml
begin: "2022-04-18T17:00:00-07:00"
```

JSON

```json
{
  "begin": "2022-04-18T17:00:00-07:00"
}
```

end

description	Date and time at which the subdue should end. In RFC 3339 format with numeric zone offset (`2022-01-01T07:30:00-07:00` or `2022-01-01T14:30:00Z`).
required	true
type	String

YML

example

```yaml
end: "2022-04-19T08:00:00-07:00"
```

JSON

```
```

```
{
  "end": "2022-04-19T08:00:00-07:00"
}
```

repeat

description	Interval at which the subdue should repeat. `weekdays` includes Mondays, Tuesdays, Wednesdays, Thursdays, and Fridays. `weekends` includes Saturdays and Sundays. Read Subdues and repeat for more information. ***NOTE**: Check subdue repeats are based on the specified `begin` and `end` times and not duration or the difference between the `begin` and `end` times.*
required	false
type	Array
allowed values	`mondays`, `tuesdays`, `wednesdays`, `thursdays`, `fridays`, `saturdays`, `sundays`, `weekdays`, `weekends`, `daily`, `weekly`, `monthly`, `annually`
example	**YML** `repeat:` `- weekdays` **JSON** `{ "repeat": [ "weekdays" ] }`

Metric check example

The following example shows the resource definition for a check that collects metrics in Nagios Performance Data format:

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  annotations:
    slack-channel: '#monitoring'
  labels:
    region: us-west-1
  name: collect-metrics
spec:
  check_hooks: null
  command: collect.sh
  discard_output: true
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: prometheus_text
  output_metric_tags:
  - name: instance
    value: '{{ .name }}'
  - name: namespace
    value: '{{ .namespace }}'
  - name: service
    value: '{{ .labels.service }}'
  output_metric_thresholds:
    - name: system_mem_used
      tags: null
      null_status: 1
      thresholds:
      - max: "75.0"
        min: ""
```

```
        status: 1
      - max: "90.0"
        min: ""
        status: 2
    - name: system_host_processes
      tags:
      - name: namespace
        value: production
      null_status: 1
      thresholds:
      - max: "50"
        min: "5"
        status: 1
      - max: "75"
        min: "2"
        status: 2
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: prometheus_gateway_workflows
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets: null
  stdin: false
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "annotations": {
      "slack-channel": "#monitoring"
    },
    "labels": {
      "region": "us-west-1"
```

```
    },
    "name": "collect-metrics"
  },
  "spec": {
    "check_hooks": null,
    "command": "collect.sh",
    "discard_output": true,
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "prometheus_text",
    "output_metric_tags": [
      {
        "name": "instance",
        "value": "{{ .name }}"
      },
      {
        "name": "namespace",
        "value": "{{ .namespace }}"
      },
      {
        "name": "service",
        "value": "{{ .labels.service }}"
      }
    ],
    "output_metric_thresholds": [
      {
        "name": "system_mem_used",
        "tags": null,
        "null_status": 1,
        "thresholds": [
          {
            "max": "75.0",
            "min": "",
            "status": 1
          },
          {
            "max": "90.0",
            "min": "",
            "status": 2
```

```
        }
      ]
    },
    {
      "name": "system_host_processes",
      "tags": [
        {
          "name": "namespace",
          "value": "production"
        }
      ],
      "null_status": 1,
      "thresholds": [
        {
          "max": "50",
          "min": "5",
          "status": 1
        },
        {
          "max": "75",
          "min": "2",
          "status": 2
        }
      ]
    }
  ],
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "prometheus_gateway_workflows"
    }
  ],
  "proxy_entity_name": "",
  "publish": true,
  "round_robin": false,
  "runtime_assets": null,
  "stdin": false,
  "subscriptions": [
    "system"
  ],
  "timeout": 0,
```

```
    "ttl": 0
  }
}
```

Check example that uses secrets management

The check in the following example uses secrets management to keep a GitHub token private. Learn more about secrets management for your Sensu configuration in the secrets and secrets providers references.

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: ping-github-api
spec:
  check_hooks: null
  command: ping-github-api.sh $GITHUB_TOKEN
  secrets:
  - name: GITHUB_TOKEN
    secret: github-token-vault
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "ping-github-api"
  },
  "spec": {
    "check_hooks": null,
    "command": "ping-github-api.sh $GITHUB_TOKEN",
    "secrets": [
      {
        "name": "GITHUB_TOKEN",
        "secret": "github-token-vault"
```

```
      }
    ]
  }
}
```

Check example with a PowerShell script command

If you use a PowerShell script in your check command, make sure to include the `-f` flag in the command. The `-f` flag ensures that the proper exit code is passed into Sensu. For example:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: interval_test
spec:
  command: powershell.exe -f c:\\users\\tester\\test.ps1
  subscriptions:
  - system
  interval: 60
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: interval_pipeline
  publish: true
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "interval_test"
  },
  "spec": {
    "command": "powershell.exe -f c:\\\\users\\ ester\\ est.ps1",
    "subscriptions": [
```

```
        "system"
      ],
      "interval": 60,
      "pipelines": [
        {
          "type": "Pipeline",
          "api_version": "core/v2",
          "name": "interval_pipeline"
        }
      ],
      "publish": true
    }
}
```

The dynamic runtime asset reference includes an example check definition that uses the asset path to correctly capture exit status codes from PowerShell plugins distributed as dynamic runtime assets.

Hooks reference

Hooks are reusable commands the agent executes in response to a check result before creating an observability event. You can create, manage, and reuse hooks independently of checks. Hooks enrich observability event context by gathering relevant information based on the exit status code of a check (ex: `1`). Hook commands can also receive JSON serialized Sensu client data via stdin.

Hook example

You can use hooks to automate data gathering for incident triage. This example demonstrates a check hook to capture the process tree when a process is not running:

YML

```
---
type: HookConfig
api_version: core/v2
metadata:
  name: process_tree
spec:
  command: ps aux
  stdin: false
  timeout: 60
  runtime_assets: null
```

JSON

```
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "process_tree"
  },
  "spec": {
    "command": "ps aux",
    "timeout": 60,
    "stdin": false,
```

```
    "runtime_assets": null
  }
}
```

Check response types

Each **type** of response (ex: `non-zero`) can contain one or more hooks and correspond to one or more exit status codes. Hooks are executed in order of precedence, based on their type:

1. `1` to `255`
2. `ok`
3. `warning`
4. `critical`
5. `unknown`
6. `non-zero`

You can assign one or more hooks to a check in the check definition. Review the check specification to configure the `check_hooks` attribute.

Check hooks

Sensu captures the hook command output, status, executed timestamp, and duration and publishes them in the resulting event.

You can use `sensuctl` to view hook command data:

SHELL

```
sensuctl event info <entity_name> <check_name> --format yaml
```

SHELL

```
sensuctl event info <entity_name> <check_name> --format wrapped-json
```

The response lists the specified event, which includes the hook command data:

YML

```yaml
---
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    ...
    hooks:
    - command: df -hT / | grep '/'
      duration: 0.002904412
      executed: 1559948435
      issued: 0
      metadata:
        name: root_disk
        namespace: default
      output: "/dev/mapper/centos-root xfs    41G  1.6G   40G   4% /\n"
      status: 0
      stdin: false
      timeout: 60
```

JSON

```json
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "...": "...",
      "hooks": [
        {
          "command": "df -hT / | grep '/'",
          "duration": 0.002904412,
          "executed": 1559948435,
          "issued": 0,
          "metadata": {
```

```
            "name": "root_disk",
            "namespace": "default"
          },
          "output": "/dev/mapper/centos-root xfs    41G  1.6G   40G   4% /\n",
          "status": 0,
          "stdin": false,
          "timeout": 60
        }
      ]
    }
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Hook specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For hooks in this version of Sensu, the `api_version` should always be `core/v2`.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{`

```
  "api_version": "core/v2"
}
```

metadata	
description	Top-level collection of metadata about the hook that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the hook definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: process_tree` `  namespace: default` `  created_by: admin` `  labels:` `  region: us-west-1` `  annotations:` `  slack-channel: "#monitoring"` **JSON** `{` `  "metadata": {` `  "name": "process_tree",` `  "namespace": "default",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1"` `  },` `  "annotations": {` `  "slack-channel": "#monitoring"`

```
    }
  }
}
```

spec

description	Top-level map that includes the hook spec attributes.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  command: ps aux` `  timeout: 60` `  stdin: false` **JSON** `{` `  "spec": {` `  "command": "ps aux",` `  "timeout": 60,` `  "stdin": false` `  }` `}`

type

description	Top-level attribute that specifies the `sensuctl create` resource type. Hooks should always be type `HookConfig`.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use

	with `sensuctl create`.
type	String
example	**YML** `type: HookConfig` **JSON** `{ "type": "HookConfig" }`

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` `  playbook: www.example.url` **JSON**

```
{
  "annotations": {
    "managed-by": "ops",
    "playbook": "www.example.url"
  }
}
```

created_by

description	Username of the Sensu user who created the hook or last updated the hook. Sensu automatically populates the `created_by` field when the hook is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.

	Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

name

description	Unique string used to identify the hook. Hook names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each hook must have a unique name within its namespace.
required	true
type	String
example	**YML** `name: process_tree`

JSON

```
{
  "name": "process_tree"
}
```

namespace	
description	The Sensu RBAC namespace that this hook belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: production` **JSON** `{ "namespace": "production" }`

Spec attributes

command	
description	Hook command to be executed.
required	true
type	String

example	YML ``` command: sudo /etc/init.d/nginx start ``` JSON ``` { "command": "sudo /etc/init.d/nginx start" } ```

runtime_assets

description	Array of Sensu dynamic runtime assets (by their names) required at runtime for execution of the `command`.
required	false
type	Array
example	YML ``` runtime_assets: - log-context ``` JSON ``` { "runtime_assets": ["log-context"] } ```

stdin

description	If `true`, the Sensu agent writes JSON serialized Sensu entity and

	check data to the command process stdin. Otherwise, `false`. The command must expect the JSON data via stdin, read it, and close stdin. This attribute cannot be used with existing Sensu check plugins or Nagios plugins because the Sensu agent will wait indefinitely for the hook process to read and close stdin.
required	false
type	Boolean
default	`false`
example	**YML** `stdin: true` **JSON** `{ "stdin": true }`

timeout

description	Hook execution duration timeout (hard stop). In seconds.
required	false
type	Integer
default	60
example	**YML** `timeout: 30` **JSON** `{ "timeout": 30 }`

Hook for rudimentary auto-remediation

You can use hooks for rudimentary auto-remediation tasks, such as starting a process that is no longer running.

> ***NOTE**: Use caution with this approach. Hooks used for auto-remediation will run without regard to the number of event occurrences.*

YML

```yaml
---
type: HookConfig
api_version: core/v2
metadata:
  name: restart_nginx
spec:
  command: sudo systemctl start nginx
  stdin: false
  timeout: 60
```

JSON

```json
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "restart_nginx"
  },
  "spec": {
    "command": "sudo systemctl start nginx",
    "timeout": 60,
    "stdin": false
  }
}
```

Hook that uses token substitution

You can create check hooks that use token substitution so you can fine-tune check attributes on a per-entity level and re-use the check definition.

> ***NOTE**: Token substitution uses entity-scoped metadata, so make sure to set labels at the entity level.*

YML

```
---
type: HookConfig
api_version: core/v2
metadata:
  labels:
    foo: bar
  name: tokensub
spec:
  command: tokensub {{ .labels.foo }}
  stdin: false
  timeout: 60
```

JSON

```
{
   "type": "HookConfig",
   "api_version": "core/v2",
   "metadata": {
      "labels": {
         "foo": "bar"
      },
      "name": "tokensub"
   },
   "spec": {
      "command": "tokensub {{ .labels.foo }}",
      "stdin": false,
      "timeout": 60
   }
}
```

Metrics reference

Sensu Go offers built-in support for collecting and processing service and time-series metrics for your entire infrastructure.

In Sensu, metrics are an optional component of observation data in events. Sensu events may contain check execution results, metrics, or both. Certain inputs like the Sensu StatsD listener or patterns like the Prometheus collector pattern will create metrics-only events. Events can also include metrics from check output metric extraction.

Use Sensu handlers to process extracted metrics and route them to databases like Elasticsearch, InfluxDB, Grafana, and Graphite. You can also use Sensu's time-series and long-term event storage integrations to process service and time-series metrics.

***NOTE**: This reference describes the metrics component of observation data included in Sensu events, which is distinct from the Sensu /metrics API. For information about HTTP GET access to internal Sensu metrics, read our /metrics API documentation.*

Metric check example

This check definition collects metrics in Graphite Plaintext Protocol format using the sensu/system-check dynamic runtime asset and sends the collected metrics to a pipeline configured with handlers that use the sensu/sensu-go-graphite-handler dynamic runtime asset:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: collect-system-metrics
spec:
  check_hooks: null
  command: system-check
  env_vars: null
  high_flap_threshold: 0
```

```
  interval: 10
  low_flap_threshold: 0
  output_metric_format: graphite_plaintext
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: graphite_workflows
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - system-check
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "collect-system-metrics"
  },
  "spec": {
    "check_hooks": null,
    "command": "system-check",
    "env_vars": null,
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "graphite_plaintext",
    "pipelines": [
      {
        "type": "Pipeline",
        "api_version": "core/v2",
        "name": "graphite_workflows"
      }
```

```
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "system-check"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

Metric event example

The example metric check will produce events similar to this metric event:

YML

```
---
pipelines:
- type: Pipeline
  api_version: core/v2
  name: graphite_workflows
timestamp: 1635270402
entity:
  entity_class: agent
  system:
    hostname: sensu-centos
    os: linux
    platform: centos
    platform_family: rhel
    platform_version: 7.5.1804
    network:
      interfaces:
```

```
      - name: lo
        addresses:
        - 127.0.0.1/8
        - "::1/128"
      - name: eth0
        mac: '08:00:27:8b:c9:3f'
        addresses:
        - 10.0.2.15/24
        - fe80::7103:bbce:3543:cfcf/64
      - name: eth1
        mac: '08:00:27:36:bb:67'
        addresses:
        - 172.28.128.89/24
        - fe80::a00:27ff:fe36:bb67/64
    arch: amd64
    libc_type: glibc
    vm_system: vbox
    vm_role: guest
    cloud_provider: ''
    processes:
  subscriptions:
  - system
  - entity:sensu-centos
  last_seen: 1635270399
  deregister: false
  deregistration: {}
  user: agent
  redact:
  - password
  - passwd
  - pass
  - api_key
  - api_token
  - access_key
  - secret_key
  - private_key
  - secret
  metadata:
    name: sensu-centos
    namespace: default
  sensu_agent_version: 6.5.1
check:
```

```
  command: system-check
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  publish: true
  runtime_assets:
  - system-check
  subscriptions:
  - system
  proxy_entity_name: ''
  check_hooks:
  stdin: false
  subdue:
  ttl: 0
  timeout: 0
  round_robin: false
  duration: 3.00889206
  executed: 1635270399
  history:
  - status: 0
    executed: 1635270359
  - status: 0
    executed: 1635270369
  - status: 0
    executed: 1635270379
  - status: 0
    executed: 1635270389
  - status: 0
    executed: 1635270399
  issued: 1635270399
  output: |+
    # HELP system_cpu_cores [GAUGE] Number of cpu cores on the system
    # TYPE system_cpu_cores GAUGE
    system_cpu_cores{} 1 1635270399219
    # HELP system_cpu_idle [GAUGE] Percent of time all cpus were idle
    # TYPE system_cpu_idle GAUGE
    system_cpu_idle{cpu="cpu0"} 99.32885906040329 1635270399219
    system_cpu_idle{cpu="cpu-total"} 99.32885906040329 1635270399219
    # HELP system_cpu_used [GAUGE] Percent of time all cpus were used
    # TYPE system_cpu_used GAUGE
    system_cpu_used{cpu="cpu0"} 0.671140939596711 1635270399219
    system_cpu_used{cpu="cpu-total"} 0.671140939596711 1635270399219
```

```
    # HELP system_cpu_user [GAUGE] Percent of time total cpu was used by normal
processes in user mode
    # TYPE system_cpu_user GAUGE
    system_cpu_user{cpu="cpu0"} 0.3355704697986485 1635270399219
    system_cpu_user{cpu="cpu-total"} 0.3355704697986485 1635270399219
    # HELP system_cpu_system [GAUGE] Percent of time all cpus used by processes
executed in kernel mode
    # TYPE system_cpu_system GAUGE
    system_cpu_system{cpu="cpu0"} 0.33557046979867833 1635270399219
    system_cpu_system{cpu="cpu-total"} 0.33557046979867833 1635270399219
    # HELP system_cpu_nice [GAUGE] Percent of time all cpus used by niced processes
in user mode
    # TYPE system_cpu_nice GAUGE
    system_cpu_nice{cpu="cpu0"} 0 1635270399219
    system_cpu_nice{cpu="cpu-total"} 0 1635270399219
    # HELP system_cpu_iowait [GAUGE] Percent of time all cpus waiting for I/O to
complete
    # TYPE system_cpu_iowait GAUGE
    system_cpu_iowait{cpu="cpu0"} 0 1635270399219
    system_cpu_iowait{cpu="cpu-total"} 0 1635270399219
    # HELP system_cpu_irq [GAUGE] Percent of time all cpus servicing interrupts
    # TYPE system_cpu_irq GAUGE
    system_cpu_irq{cpu="cpu0"} 0 1635270399219
    system_cpu_irq{cpu="cpu-total"} 0 1635270399219
    # HELP system_cpu_sortirq [GAUGE] Percent of time all cpus servicing software
interrupts
    # TYPE system_cpu_sortirq GAUGE
    system_cpu_sortirq{cpu="cpu0"} 0 1635270399219
    system_cpu_sortirq{cpu="cpu-total"} 0 1635270399219
    # HELP system_cpu_stolen [GAUGE] Percent of time all cpus serviced virtual hosts
operating systems
    # TYPE system_cpu_stolen GAUGE
    system_cpu_stolen{cpu="cpu0"} 0 1635270399219
    system_cpu_stolen{cpu="cpu-total"} 0 1635270399219
    # HELP system_cpu_guest [GAUGE] Percent of time all cpus serviced guest
operating system
    # TYPE system_cpu_guest GAUGE
    system_cpu_guest{cpu="cpu0"} 0 1635270399219
    system_cpu_guest{cpu="cpu-total"} 0 1635270399219
    # HELP system_cpu_guest_nice [GAUGE] Percent of time all cpus serviced niced
guest operating system
    # TYPE system_cpu_guest_nice GAUGE
```

```
    system_cpu_guest_nice{cpu="cpu0"} 0 1635270399219
    system_cpu_guest_nice{cpu="cpu-total"} 0 1635270399219
    # HELP system_mem_used [GAUGE] Percent of memory used
    # TYPE system_mem_used GAUGE
    system_mem_used{} 21.21448463577672 1635270399219
    # HELP system_mem_used_bytes [GAUGE] Used memory in bytes
    # TYPE system_mem_used_bytes GAUGE
    system_mem_used_bytes{} 2.20598272e+08 1635270399219
    # HELP system_mem_total_bytes [GAUGE] Total memory in bytes
    # TYPE system_mem_total_bytes GAUGE
    system_mem_total_bytes{} 1.039847424e+09 1635270399219
    # HELP system_swap_used [GAUGE] Percent of swap used
    # TYPE system_swap_used GAUGE
    system_swap_used{} 0 1635270399219
    # HELP system_swap_used_bytes [GAUGE] Used swap in bytes
    # TYPE system_swap_used_bytes GAUGE
    system_swap_used_bytes{} 2.20598272e+08 1635270399219
    # HELP system_swap_total_bytes [GAUGE] Total swap in bytes
    # TYPE system_swap_total_bytes GAUGE
    system_swap_total_bytes{} 2.147479552e+09 1635270399219
    # HELP system_load_load1 [GAUGE] System load averaged over 1 minute, high load
value dependant on number of cpus in system
    # TYPE system_load_load1 GAUGE
    system_load_load1{} 0 1635270399219
    # HELP system_load_load5 [GAUGE] System load averaged over 5 minute, high load
value dependent on number of cpus in system
    # TYPE system_load_load5 GAUGE
    system_load_load5{} 0.01 1635270399219
    # HELP system_load_load15 [GAUGE] System load averaged over 15 minute, high load
value dependent on number of cpus in system
    # TYPE system_load_load15 GAUGE
    system_load_load15{} 0.05 1635270399219
    # HELP system_load_load1_per_cpu [GAUGE] System load averaged over 1 minute
normalized by cpu count, values \u003e 1 means system may be overloaded
    # TYPE system_load_load1_per_cpu GAUGE
    system_load_load1_per_cpu{} 0 1635270399219
    # HELP system_load_load5_per_cpu [GAUGE] System load averaged over 5 minute
normalized by cpu count, values \u003e 1 means system may be overloaded
    # TYPE system_load_load5_per_cpu GAUGE
    system_load_load5_per_cpu{} 0.01 1635270399219
    # HELP system_load_load15_per_cpu [GAUGE] System load averaged over 15 minute
normalized by cpu count, values \u003e 1 means system may be overloaded
```

```
    # TYPE system_load_load15_per_cpu GAUGE
    system_load_load15_per_cpu{} 0.05 1635270399219
    # HELP system_host_uptime [COUNTER] Host uptime in seconds
    # TYPE system_host_uptime COUNTER
    system_host_uptime{} 982 1635270399219
    # HELP system_host_processes [GAUGE] Number of host processes
    # TYPE system_host_processes GAUGE
    system_host_processes{} 109 1635270399219
  state: passing
  status: 0
  total_state_change: 0
  last_ok: 1635270399
  occurrences: 5
  occurrences_watermark: 5
  output_metric_format: graphite_plaintext
  env_vars:
  metadata:
    name: collect-system-metrics
    namespace: default
  secrets:
  is_silenced: false
  scheduler: memory
  processed_by: sensu-centos
metrics:
  points:
  - name: system_cpu_cores{}
    value: 1
    timestamp: 1635270399219
    tags:
  - name: system_cpu_idle{cpu="cpu0"}
    value: 99.32885906040329
    timestamp: 1635270399219
    tags:
  - name: system_cpu_idle{cpu="cpu-total"}
    value: 99.32885906040329
    timestamp: 1635270399219
    tags:
  - name: system_cpu_used{cpu="cpu0"}
    value: 0.671140939596711
    timestamp: 1635270399219
    tags:
  - name: system_cpu_used{cpu="cpu-total"}
```

```
    value: 0.671140939596711
    timestamp: 1635270399219
    tags:
  - name: system_cpu_user{cpu="cpu0"}
    value: 0.3355704697986485
    timestamp: 1635270399219
    tags:
  - name: system_cpu_user{cpu="cpu-total"}
    value: 0.3355704697986485
    timestamp: 1635270399219
    tags:
  - name: system_cpu_system{cpu="cpu0"}
    value: 0.33557046979867833
    timestamp: 1635270399219
    tags:
  - name: system_cpu_system{cpu="cpu-total"}
    value: 0.33557046979867833
    timestamp: 1635270399219
    tags:
  - name: system_cpu_nice{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_nice{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_iowait{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_iowait{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_irq{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_irq{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
```

```
    tags:
  - name: system_cpu_sortirq{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_sortirq{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_stolen{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_stolen{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_guest{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_guest{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_guest_nice{cpu="cpu0"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_cpu_guest_nice{cpu="cpu-total"}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_mem_used{}
    value: 21.21448463577672
    timestamp: 1635270399219
    tags:
  - name: system_mem_used_bytes{}
    value: 220598272
    timestamp: 1635270399219
    tags:
  - name: system_mem_total_bytes{}
```

```
    value: 1039847424
    timestamp: 1635270399219
    tags:
  - name: system_swap_used{}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_swap_used_bytes{}
    value: 220598272
    timestamp: 1635270399219
    tags:
  - name: system_swap_total_bytes{}
    value: 2147479552
    timestamp: 1635270399219
    tags:
  - name: system_load_load1{}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_load_load5{}
    value: 0.01
    timestamp: 1635270399219
    tags:
  - name: system_load_load15{}
    value: 0.05
    timestamp: 1635270399219
    tags:
  - name: system_load_load1_per_cpu{}
    value: 0
    timestamp: 1635270399219
    tags:
  - name: system_load_load5_per_cpu{}
    value: 0.01
    timestamp: 1635270399219
    tags:
  - name: system_load_load15_per_cpu{}
    value: 0.05
    timestamp: 1635270399219
    tags:
  - name: system_host_uptime{}
    value: 982
    timestamp: 1635270399219
```

```
    tags:
  - name: system_host_processes{}
    value: 109
    timestamp: 1635270399219
    tags:
metadata:
  namespace: default
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
sequence: 5
```

JSON

```
{
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "graphite_workflows"
    }
  ],
  "timestamp": 1635270402,
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
```

```
            "10.0.2.15/24",
            "fe80::7103:bbce:3543:cfcf/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:36:bb:67",
          "addresses": [
            "172.28.128.89/24",
            "fe80::a00:27ff:fe36:bb67/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "system",
    "entity:sensu-centos"
  ],
  "last_seen": 1635270399,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
    "name": "sensu-centos",
```

```
      "namespace": "default"
    },
    "sensu_agent_version": "6.5.1"
  },
  "check": {
    "command": "system-check",
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "system-check"
    ],
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 3.00889206,
    "executed": 1635270399,
    "history": [
      {
        "status": 0,
        "executed": 1635270359
      },
      {
        "status": 0,
        "executed": 1635270369
      },
      {
        "status": 0,
        "executed": 1635270379
      },
      {
        "status": 0,
        "executed": 1635270389
      },
```

```
      {
        "status": 0,
        "executed": 1635270399
      }
    ],
    "issued": 1635270399,
    "output": "# HELP system_cpu_cores [GAUGE] Number of cpu cores on the system\n#
TYPE system_cpu_cores GAUGE\nsystem_cpu_cores{} 1 1635270399219\n# HELP
system_cpu_idle [GAUGE] Percent of time all cpus were idle\n# TYPE system_cpu_idle
GAUGE\nsystem_cpu_idle{cpu=\"cpu0\"} 99.32885906040329
1635270399219\nsystem_cpu_idle{cpu=\"cpu-total\"} 99.32885906040329 1635270399219\n#
HELP system_cpu_used [GAUGE] Percent of time all cpus were used\n# TYPE
system_cpu_used GAUGE\nsystem_cpu_used{cpu=\"cpu0\"} 0.671140939596711
1635270399219\nsystem_cpu_used{cpu=\"cpu-total\"} 0.671140939596711 1635270399219\n#
HELP system_cpu_user [GAUGE] Percent of time total cpu was used by normal processes
in user mode\n# TYPE system_cpu_user GAUGE\nsystem_cpu_user{cpu=\"cpu0\"}
0.3355704697986485 1635270399219\nsystem_cpu_user{cpu=\"cpu-total\"}
0.3355704697986485 1635270399219\n# HELP system_cpu_system [GAUGE] Percent of time
all cpus used by processes executed in kernel mode\n# TYPE system_cpu_system
GAUGE\nsystem_cpu_system{cpu=\"cpu0\"} 0.33557046979867833
1635270399219\nsystem_cpu_system{cpu=\"cpu-total\"} 0.33557046979867833
1635270399219\n# HELP system_cpu_nice [GAUGE] Percent of time all cpus used by niced
processes in user mode\n# TYPE system_cpu_nice GAUGE\nsystem_cpu_nice{cpu=\"cpu0\"}
0 1635270399219\nsystem_cpu_nice{cpu=\"cpu-total\"} 0 1635270399219\n# HELP
system_cpu_iowait [GAUGE] Percent of time all cpus waiting for I/O to complete\n#
TYPE system_cpu_iowait GAUGE\nsystem_cpu_iowait{cpu=\"cpu0\"} 0
1635270399219\nsystem_cpu_iowait{cpu=\"cpu-total\"} 0 1635270399219\n# HELP
system_cpu_irq [GAUGE] Percent of time all cpus servicing interrupts\n# TYPE
system_cpu_irq GAUGE\nsystem_cpu_irq{cpu=\"cpu0\"} 0
1635270399219\nsystem_cpu_irq{cpu=\"cpu-total\"} 0 1635270399219\n# HELP
system_cpu_sortirq [GAUGE] Percent of time all cpus servicing software interrupts\n#
TYPE system_cpu_sortirq GAUGE\nsystem_cpu_sortirq{cpu=\"cpu0\"} 0
1635270399219\nsystem_cpu_sortirq{cpu=\"cpu-total\"} 0 1635270399219\n# HELP
system_cpu_stolen [GAUGE] Percent of time all cpus serviced virtual hosts operating
systems\n# TYPE system_cpu_stolen GAUGE\nsystem_cpu_stolen{cpu=\"cpu0\"} 0
1635270399219\nsystem_cpu_stolen{cpu=\"cpu-total\"} 0 1635270399219\n# HELP
system_cpu_guest [GAUGE] Percent of time all cpus serviced guest operating system\n#
TYPE system_cpu_guest GAUGE\nsystem_cpu_guest{cpu=\"cpu0\"} 0
1635270399219\nsystem_cpu_guest{cpu=\"cpu-total\"} 0 1635270399219\n# HELP
system_cpu_guest_nice [GAUGE] Percent of time all cpus serviced niced guest
operating system\n# TYPE system_cpu_guest_nice
GAUGE\nsystem_cpu_guest_nice{cpu=\"cpu0\"} 0
```

1635270399219\nsystem_cpu_guest_nice{cpu=\"cpu-total\"} 0 1635270399219\n# HELP system_mem_used [GAUGE] Percent of memory used\n# TYPE system_mem_used GAUGE\nsystem_mem_used{} 21.21448463577672 1635270399219\n# HELP system_mem_used_bytes [GAUGE] Used memory in bytes\n# TYPE system_mem_used_bytes GAUGE\nsystem_mem_used_bytes{} 2.20598272e+08 1635270399219\n# HELP system_mem_total_bytes [GAUGE] Total memory in bytes\n# TYPE system_mem_total_bytes GAUGE\nsystem_mem_total_bytes{} 1.039847424e+09 1635270399219\n# HELP system_swap_used [GAUGE] Percent of swap used\n# TYPE system_swap_used GAUGE\nsystem_swap_used{} 0 1635270399219\n# HELP system_swap_used_bytes [GAUGE] Used swap in bytes\n# TYPE system_swap_used_bytes GAUGE\nsystem_swap_used_bytes{} 2.20598272e+08 1635270399219\n# HELP system_swap_total_bytes [GAUGE] Total swap in bytes\n# TYPE system_swap_total_bytes GAUGE\nsystem_swap_total_bytes{} 2.147479552e+09 1635270399219\n# HELP system_load_load1 [GAUGE] System load averaged over 1 minute, high load value dependant on number of cpus in system\n# TYPE system_load_load1 GAUGE\nsystem_load_load1{} 0 1635270399219\n# HELP system_load_load5 [GAUGE] System load averaged over 5 minute, high load value dependent on number of cpus in system\n# TYPE system_load_load5 GAUGE\nsystem_load_load5{} 0.01 1635270399219\n# HELP system_load_load15 [GAUGE] System load averaged over 15 minute, high load value dependent on number of cpus in system\n# TYPE system_load_load15 GAUGE\nsystem_load_load15{} 0.05 1635270399219\n# HELP system_load_load1_per_cpu [GAUGE] System load averaged over 1 minute normalized by cpu count, values \\u003e 1 means system may be overloaded\n# TYPE system_load_load1_per_cpu GAUGE\nsystem_load_load1_per_cpu{} 0 1635270399219\n# HELP system_load_load5_per_cpu [GAUGE] System load averaged over 5 minute normalized by cpu count, values \\u003e 1 means system may be overloaded\n# TYPE system_load_load5_per_cpu GAUGE\nsystem_load_load5_per_cpu{} 0.01 1635270399219\n# HELP system_load_load15_per_cpu [GAUGE] System load averaged over 15 minute normalized by cpu count, values \\u003e 1 means system may be overloaded\n# TYPE system_load_load15_per_cpu GAUGE\nsystem_load_load15_per_cpu{} 0.05 1635270399219\n# HELP system_host_uptime [COUNTER] Host uptime in seconds\n# TYPE system_host_uptime COUNTER\nsystem_host_uptime{} 982 1635270399219\n# HELP system_host_processes [GAUGE] Number of host processes\n# TYPE system_host_processes GAUGE\nsystem_host_processes{} 109 1635270399219\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1635270399,
    "occurrences": 5,
    "occurrences_watermark": 5,
    "output_metric_format": "graphite_plaintext",
    "env_vars": null,
    "metadata": {

```
      "name": "collect-system-metrics",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "scheduler": "memory",
    "processed_by": "sensu-centos"
  },
  "metrics": {
    "points": [
      {
        "name": "system_cpu_cores{}",
        "value": 1,
        "timestamp": 1635270399219,
        "tags": null
      },
      {
        "name": "system_cpu_idle{cpu=\"cpu0\"}",
        "value": 99.32885906040329,
        "timestamp": 1635270399219,
        "tags": null
      },
      {
        "name": "system_cpu_idle{cpu=\"cpu-total\"}",
        "value": 99.32885906040329,
        "timestamp": 1635270399219,
        "tags": null
      },
      {
        "name": "system_cpu_used{cpu=\"cpu0\"}",
        "value": 0.671140939596711,
        "timestamp": 1635270399219,
        "tags": null
      },
      {
        "name": "system_cpu_used{cpu=\"cpu-total\"}",
        "value": 0.671140939596711,
        "timestamp": 1635270399219,
        "tags": null
      },
      {
        "name": "system_cpu_user{cpu=\"cpu0\"}",
```

```
      "value": 0.3355704697986485,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_user{cpu=\"cpu-total\"}",
      "value": 0.3355704697986485,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_system{cpu=\"cpu0\"}",
      "value": 0.33557046979867833,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_system{cpu=\"cpu-total\"}",
      "value": 0.33557046979867833,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_nice{cpu=\"cpu0\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_nice{cpu=\"cpu-total\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_iowait{cpu=\"cpu0\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_iowait{cpu=\"cpu-total\"}",
```

```
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_irq{cpu=\"cpu0\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_irq{cpu=\"cpu-total\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_sortirq{cpu=\"cpu0\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_sortirq{cpu=\"cpu-total\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_stolen{cpu=\"cpu0\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_stolen{cpu=\"cpu-total\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_guest{cpu=\"cpu0\"}",
```

```
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_guest{cpu=\"cpu-total\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_guest_nice{cpu=\"cpu0\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_cpu_guest_nice{cpu=\"cpu-total\"}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_mem_used{}",
      "value": 21.21448463577672,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_mem_used_bytes{}",
      "value": 220598272,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_mem_total_bytes{}",
      "value": 1039847424,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_swap_used{}",
```

```
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_swap_used_bytes{}",
      "value": 220598272,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_swap_total_bytes{}",
      "value": 2147479552,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_load_load1{}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_load_load5{}",
      "value": 0.01,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_load_load15{}",
      "value": 0.05,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_load_load1_per_cpu{}",
      "value": 0,
      "timestamp": 1635270399219,
      "tags": null
    },
    {
      "name": "system_load_load5_per_cpu{}",
```

```
          "value": 0.01,
          "timestamp": 1635270399219,
          "tags": null
        },
        {
          "name": "system_load_load15_per_cpu{}",
          "value": 0.05,
          "timestamp": 1635270399219,
          "tags": null
        },
        {
          "name": "system_host_uptime{}",
          "value": 982,
          "timestamp": 1635270399219,
          "tags": null
        },
        {
          "name": "system_host_processes{}",
          "value": 109,
          "timestamp": 1635270399219,
          "tags": null
        }
      ]
    },
    "metadata": {
      "namespace": "default"
    },
    "id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "sequence": 5
}
```


***NOTE**: Metrics data points are not included in events retrieved with `sensuctl event info` — these events include check output text rather than a set of metrics points. To view metrics points data, add a debug handler that prints events to a JSON file.*

Extract metrics from check output

The Sensu agent can extract metrics data from check command output and populate an event's

metrics attribute before sending the event to the Sensu backend for processing.

To extract metrics from check output:

- The check `command` execution must output metrics in one of Sensu's supported output metric formats.
- The check must include the `output_metric_format` attribute with a value that specifies one of Sensu's supported output metric formats.

When a check includes correctly configured `command` and `output_metric_format` attributes, Sensu will extract the specified metrics from the check output and add them to the event data in the metrics attribute.

Supported output metric formats

Sensu supports the following formats for check output metric extraction.

Graphite	
output metric format	`graphite_plaintext`
documentation	Graphite Plaintext Protocol
example	`local.random.diceroll 4 123456789`

InfluxDB	
output metric format	`influxdb_line`
documentation	InfluxDB Line Protocol
example	`weather,location=us-midwest temperature=82 1465839830100400200`

Nagios	
output metric format	`nagios_perfdata`
documentation	Nagios Performance Data
example	`PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80`

OpenTSDB	
output metric format	`opentsdb_line`
documentation	OpenTSDB Data Specification
example	`sys.cpu.user 1356998400 42.5 host=webserver01 cpu=0`

Prometheus	
output metric format	`prometheus_text`
documentation	Prometheus Exposition Text
example	`http_requests_total{method="post",code="200"} 1027 1395066363000`

Enrich metrics with tags

In metric check output, metrics data points include the `tags` array. Tags add information for the metrics points in events. For example, a tag can specify the name of the check or entity associated with a specific metrics point.

Tags can be generated in various ways, like plugin code or a third-party exporter. You can also add specific tags to metrics points with output metric tags.

Add output metric tags

Output metric tags are custom tags you can add to your check definition to enrich the metrics data points produced by check output metric extraction with additional context.

The key-value pairs you add to a check's `output_metric_tags` array will be included in the `tags` array after check output metric extraction. For example, suppose you include this `output_metric_tags` array in your check:

YML

```
output_metric_tags:
- name: instance
  value: sensu-centos-1
- name: prometheus_type
  value: gauge
```

JSON

```
{
  "output_metric_tags": [
    {
      "name": "instance",
      "value": "sensu-centos-1"
    },
    {
      "name": "prometheus_type",
      "value": "gauge"
    }
  ]
}
```

In check output, the metrics points would include the output metric tags in the `tags` array, similar to this example:

TEXT

```
points:
- name: dns_duration
  value: 0.000251
  timestamp: 1648220984
  tags:
  - name: instance
    value: sensu-centos-1
  - name: prometheus_type
    value: gauge
- name: tls_handshake_duration
  value: 0
  timestamp: 1648220984
  tags:
  - name: instance
    value: sensu-centos-1
  - name: prometheus_type
    value: gauge
```

TEXT

```
{
  "points": [
    {
      "name": "dns_duration",
      "value": 0.000251,
      "timestamp": 1648220984,
      "tags": [
        {
          "name": "instance",
          "value": "sensu-centos-1"
        },
        {
          "name": "prometheus_type",
          "value": "gauge"
        }
      ]
    },
    {
      "name": "tls_handshake_duration",
      "value": 0,
      "timestamp": 1648220984,
```

```
      "tags": [
        {
          "name": "instance",
          "value": "sensu-centos-1"
        },
        {
          "name": "prometheus_type",
          "value": "gauge"
        }
      ]
    }
  ]
}
```

Sensu adds any output metric tag values to the `tags` array along with any natively supported tags produced by check output metric extraction.

Use token substitution with output metric tags

Use token substitution to include any event attribute in an output metric tag. Add token substitution in the output metric tag `value` attribute. For example, these tags will list the `event.timestamp` and `event.entity.name` attributes:

YML

```
---
output_metric_tags:
- name: time
  value: "{{ .timestamp }}"
- name: entity_name
  value: "{{ .entity.name }}"
```

JSON

```
{
  "output_metric_tags": [
    {
      "name": "time",
      "value": "{{ .timestamp }}"
```

```
    },
    {
      "name": "entity_name",
      "value": "{{ .entity.name }}"
    }
  ]
}
```

Collect metrics in formats that do not support tags

Output metric tags are useful when you want to collect metrics in a format that does not natively support tags, like Nagios Performance Data.

For example, you might want to collect and transmit metrics in Nagios Performance Data format, which does not support tags, and store the metrics in Prometheus, which does support tags. In this case, you can specify the tags to include with metrics with output metric tags. The `output_metric_format`, `output_metric_handlers`, and `output_metric_tags` attributes in your check definition might look similar to this example:

YML

```
output_metric_format: nagios_perfdata
output_metric_handlers:
  - prometheus_gateway
output_metric_tags:
  - name: instance
    value: '{{ .name }}'
  - name: prometheus_type
    value: gauge
  - name: service
    value: '{{ .labels.service }}'
```

JSON

```
{
  "output_metric_format": "nagios_perfdata",
  "output_metric_handlers": [
    "prometheus_gateway"
  ],
  "output_metric_tags": [
```

```
    {
      "name": "instance",
      "value": "{{ .name }}"
    },
    {
      "name": "prometheus_type",
      "value": "gauge"
    },
    {
      "name": "service",
      "value": "{{ .labels.service }}"
    }
  ]
}
```

Metric threshold evaluation

Metric threshold evaluation extends Sensu's service check and metrics processing capabilities so you can get real-time alerts based on the metrics your Sensu checks collect. The Sensu agent analyzes output metrics against the thresholds you specify and overrides the event check status if the metrics values exceed the threshold values.

For example, the check from the Sensu Plus guide uses the sensu/system-check dynamic runtime asset to collect baseline system metrics. Add the `output_metric_thresholds` array to get alerts based on the Sensu System Check metrics `system_mem_used` (percent of memory used) and `system_host_processes` (number of host processes):

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: system-check
spec:
  command: system-check
  runtime_assets:
  - system-check
  subscriptions:
  - system
```

```
interval: 10
timeout: 5
publish: true
pipelines:
- type: Pipeline
  api_version: core/v2
  name: sensu_to_sumo
output_metric_format: prometheus_text
output_metric_tags:
- name: entity
  value: "{{ .name }}"
- name: namespace
  value: "{{ .namespace }}"
- name: os
  value: "{{ .system.os }}"
- name: platform
  value: "{{ .system.platform }}"
output_metric_thresholds:
- name: system_mem_used
  tags:
  null_status: 1
  thresholds:
  - max: '75.0'
    min: ''
    status: 1
  - max: '90.0'
    min: ''
    status: 2
- name: system_host_processes
  tags:
  - name: namespace
    value: production
  null_status: 1
  thresholds:
  - max: '50'
    min: '5'
    status: 1
  - max: '75'
    min: '2'
    status: 2
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "system-check"
  },
  "spec": {
    "command": "system-check",
    "runtime_assets": [
      "system-check"
    ],
    "subscriptions": [
      "system"
    ],
    "interval": 10,
    "timeout": 5,
    "publish": true,
    "pipelines": [
      {
        "type": "Pipeline",
        "api_version": "core/v2",
        "name": "sensu_to_sumo"
      }
    ],
    "output_metric_format": "prometheus_text",
    "output_metric_tags": [
      {
        "name": "entity",
        "value": "{{ .name }}"
      },
      {
        "name": "namespace",
        "value": "{{ .namespace }}"
      },
      {
        "name": "os",
        "value": "{{ .system.os }}"
      },
      {
        "name": "platform",
        "value": "{{ .system.platform }}"
```

```
    }
  ],
  "output_metric_thresholds": [
    {
      "name": "system_mem_used",
      "tags": null,
      "null_status": 1,
      "thresholds": [
        {
          "max": "75.0",
          "min": "",
          "status": 1
        },
        {
          "max": "90.0",
          "min": "",
          "status": 2
        }
      ]
    },
    {
      "name": "system_host_processes",
      "tags": [
        {
          "name": "namespace",
          "value": "production"
        }
      ],
      "null_status": 1,
      "thresholds": [
        {
          "max": "50",
          "min": "5",
          "status": 1
        },
        {
          "max": "75",
          "min": "2",
          "status": 2
        }
      ]
    }
```

```
      ]
    }
  }
```

In this example, for both `system_mem_used` and `system_host_processes`, Sensu will compare the output metrics in each event with the thresholds set for each metric. If the output metrics match or exceed the thresholds, Sensu will override the check status.

For `system_mem_used`:

- Set event status to `1` (warning) if the output metrics do not include `system_mem_used`.
- Set event status to `1` (warning) when 75% of memory is used.
- Set event status to `2` (critical) when 90% of memory is used.

For `system_host_processes`:

- Evaluate only output metrics for entities whose tags include `name: namespace` and `value: production`.
- Set event status to `1` (warning) if the output metrics do not include `system_host_processes`.
- Set event status to `1` (warning) when the number of host processes reaches 50 or more or 5 or fewer.
- Set event status to `2` (critical) when the number of host processes reaches 75 or more or 2 or fewer.

***NOTE**: The Sensu Plus example handler processes and transmits metrics data but cannot send alerts. Read Send data to Sumo Logic with Sensu to create a handler that sends alerts to Sumo Logic, which you can add to the Sensu Plus example pipeline.*

Metric threshold evaluation takes place *after* Sensu extracts metrics and *before* Sensu processes any check hooks. If you specify a metric name and tags that match more than one check output metric point, Sensu evaluates all matching metric points against the thresholds.

Check configuration requirements for metric threshold evaluation

To apply metric threshold evaluation, check definitions must include:

- The `output_metric_format` attribute with a value that specifies one of Sensu's supported output metric formats.
- The `output_metric_thresholds` array, with values specified for `name` and `thresholds`.

In addition, check status must be 0 (OK), indicating that Sensu successfully collected metrics, for the Sensu agent to evaluate the collected metrics against the specified thresholds.

Use token substitution in thresholds values

You can use check token substitution in values for `thresholds` `max` and `min` attributes instead of specifying a single constant value.Check tokens are placeholders that the Sensu agent will replace with the corresponding entity definition attribute values.

This example shows the `thresholds` array configured to use token substitution for the `max` and `min` attribute values:

YML

```
thresholds:
- max: '{{ .annotations.system_cpu_used_warning_threshold | default "70.0" }}'
  min: '{{ .annotations.system_cpu_used_warning_threshold | default "50.0" }}'
  status: 1
- max: '{{ .annotations.system_cpu_used_warning_threshold | default "80.0" }}'
  min: '{{ .annotations.system_cpu_used_warning_threshold | default "40.0" }}'
  status: 2
```

JSON

```
{
  "thresholds": [
    {
      "max": "{{ .annotations.system_cpu_used_warning_threshold | default \"70.0\"
}}",
      "min": "{{ .annotations.system_cpu_used_warning_threshold | default \"50.0\"
}}",
      "status": 1
    },
    {
      "max": "{{ .annotations.system_cpu_used_warning_threshold | default \"80.0\"
}}",
```

```
      "min": "{{ .annotations.system_cpu_used_warning_threshold | default \"40.0\"
}}",
      "status": 2
    }
  ]
}
```

If an entity has an annotation that matches `system_cpu_used_warning_threshold`, the check will substitute the annotation value when executing the check.If an entity does not have a matching annotation, the check will use the specified default values instead.

Add event annotations based on metric threshold evaluation

If a check definition includes the `output_metric_thresholds` attribute, the check's metric events with non-zero status will include an annotation that lists the reason for the status.Sensu adds one annotation per matched threshold rule, one annotation per missing metric (`null_status`), and one annotation that lists the global status for the check.

Annotations based on specified threshold values are similar to this example:

TEXT

```
annotations:
  sensu.io/output_metric_thresholds/system_mem_used/min/critical: 'The value of
"system_mem_used" exceeded the configured threshold (max: 90, actual: 95)'
```

TEXT

```
{
  "annotations": {
    "sensu.io/output_metric_thresholds/system_mem_used/min/critical": "The value of
\"system_mem_used\" exceeded the configured threshold (max: 90, actual: 95)"
  }
}
```

Annotations based on `null_status` are similar to this example:

TEXT

```
annotations:
  sensu.io/output_metric_thresholds/system_host_processes/null: 'WARNING: no metric
matching "system_host_processes" (namespace="production") was found; expected min: 5
- max: 50 (status: warning) min:2 - max: 75 (status: critical)'
```

TEXT

```
{
  "annotations": {
    "sensu.io/output_metric_thresholds/system_host_processes/null": "WARNING: no
metric matching \"system_host_processes\" (namespace=\"production\") was found;
expected min: 5 - max: 50 (status: warning) min:2 - max: 75 (status: critical)"
  }
}
```

Annotations based on global status for the check are similar to this example:

TEXT

```
annotations:
  sensu.io/notifications/critical: 'The value of node_load1 exceeded the configured
threshold (max: 4.0, actual: 5.263671875).'
```

TEXT

```
{
  "annotations": {
    "sensu.io/notifications/critical": "The value of node_load1 exceeded the
configured threshold (max: 4.0, actual: 5.263671875)."
  }
}
```

Annotations based on global `null_status` for the check are similar to this example:

TEXT

```
annotations:
  sensu.io/notifications/unknown: 'WARNING: no metric matching "node_load1"
```

```
(namespace="production") was found; expected min: 4.0 (status: warning); expected
max: 6 (status: critical)'
```

TEXT

```
{
  "annotations": {
    "sensu.io/notifications/unknown": "WARNING: no metric matching \"node_load1\"
(namespace=\"production\") was found; expected min: 4.0 (status: warning); expected
max: 6 (status: critical)"
  }
}
```

Process extracted and tagged metrics

Specify the handlers you want to process your Sensu metrics in a pipeline, then reference the pipeline in the check `pipelines` array. With handlers, you can route metrics to one or more databases for storing and visualizing metrics, like Elasticsearch, InfluxDB, Grafana, and Graphite.

Many of our most popular metrics integrations for time-series and long-term event storage include curated, configurable quick-start templates to integrate Sensu with your existing workflows. Use Bonsai, the Sensu asset hub, to discover, download, and share dynamic runtime assets for processing metrics.

To handle both metrics and status events without applying conditional filter logic, configure a pipeline with different workflows for metrics and status. The events reference includes an example event with check and metric data. Read the pipelines reference for more information about configuring a pipeline with multiple workflows.

You do not need to add a mutator to your check definition to process metrics with an event handler. The metrics attribute format automatically reduces metrics data complexity so event handlers can process metrics effectively.

Validate metrics

If the check output is formatted correctly according to its `output_metric_format`, the metrics will be extracted in Sensu metric format and passed to the observability pipeline. The Sensu agent will log errors if it cannot parse the check output.

Use the debug handler example to write metric events to a file for inspection. To confirm that the check extracted metrics, inspect the event passed to the handler in the debug-event.json file. The event will include a top-level metrics section populated with metrics points arrays if the Sensu agent correctly ingested the metrics.

Metrics specification

The check specification describes metrics attributes in checks.

The event specification describes metrics attributes in events.

Rule templates reference

***COMMERCIAL FEATURE**: Access business service monitoring (BSM), including rule templates, in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

Rule templates are the resources that Sensu applies to service components for business service monitoring (BSM). A rule template applies to selections of events defined by a service component's query. This selection of events is the rule's input.

The rule template evaluates the selection of events using an ECMAScript 5 (JavaScript) expression specified in the rule template's `eval` object and emits a single event based on this evaluation. For example, a rule template's expression might define the thresholds at which Sensu will consider a service component online, degraded, or offline:

- Online until fewer than 70% of the service component's events have a check status of OK.
- Degraded while 50-69% of the service component's events have a check status of OK.
- Offline when fewer than 50% of the service component's events have a check status of OK.

The rule template expression can also create arbitrary events.

Built-in rule template: Aggregate

Sensu includes a built-in rule template, `aggregate`, that allows you to treat the results of multiple disparate check executions executed across multiple disparate systems as a single result (event). This built-in rule template is ready to use with your service components.

Reference the rule template name in the `rules.template` field and configure the arguments in the `rules.template.arguments` object in your service component resource definitions.

Use the `aggregate` rule template for services that can be considered healthy as long as a minimum threshold is satisfied. For example, you might set the minimum threshold at 10 web servers with an OK

status or 70% of processes running with an OK status.

The `aggregate` rule template is useful in dynamic environments and environments with some tolerance for failure.

To review the `aggregate` resource definition, retrieve it with a GET request to /enterprise/bsm/v1:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates/aggregate \
-H "Authorization: Key $SENSU_API_KEY"
```

The response will include the complete `aggregate` rule template resource definition in JSON format:

```
{
  "type": "RuleTemplate",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "aggregate"
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical status if there the number of critical events is equal to or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a critical status if the percentage of non-zero events is equal to or greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers to use for produced metrics",
          "items": {
            "type": "string"
          },
```

```
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from aggregate data and include them in
the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a warning status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event with a warning status if the percentage of
non-zero events is equal to or greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service - aggregate one or more events
into a single event. This BSM rule template allows you to treat the results of
multiple disparate check executions - executed across multiple disparate systems -
as a single event. This template is extremely useful in dynamic environments and/or
environments that have a reasonable tolerance for failure. Use this template when a
service can be considered healthy as long as a minimum threshold is satisfied (for
example, at least 5 healthy web servers? at least 70% of N processes healthy?).",
    "eval": "\nif (events && events.length == 0) {\n    event.check.output =
\"WARNING: No events selected for aggregate\n\";\n    event.check.status = 1;\n
return event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_event_count\"] =
events.length;\n\npercentOK = sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var handlers = [];\n\n    if
(!!args[\"metric_handlers\"]) {\n        handlers =
args[\"metric_handlers\"].slice();\n    }\n\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n    event.timestamp = ts;\n\n    var tags = [\n
{\n            name: \"service\",\n            value: event.entity.name\n
```

```
},\n        {\n            name: \"entity\",\n            value: event.entity.name\n
},\n        {\n            name: \"check\",\n            value: event.check.name\n
}\n    ];\n\n    event.metrics = sensu.NewMetrics({\n        handlers: handlers,\n
points: [\n            {\n                name: \"percent_non_zero\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n                name: \"percent_ok\",\n
timestamp: ts,\n                value: percentOK,\n                tags: tags\n
},\n            {\n                name: \"percent_warning\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n                name:
\"percent_critical\",\n                timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n                tags: tags\n
},\n            {\n                name: \"percent_unknown\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n                name:
\"count_non_zero\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"non-zero\"),\n                tags: tags\n            },\n
{\n                name: \"count_ok\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"ok\"),\n                tags: tags\n            },\n
{\n                name: \"count_warning\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"warning\"),\n                tags: tags\n
},\n            {\n                name: \"count_critical\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n                name:
\"count_unknown\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n                tags: tags\n            }\n
]\n    });\n\n    if (!!args[\"set_metric_annotations\"]) {\n        var i = 0;\n\n
while(i < event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n            i++;\n        }\n
}\n}\n\nif (!!args[\"critical_threshold\"] && percentOK <=
args[\"critical_threshold\"]) {\n    event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] && percentOK <=
args[\"warning_threshold\"]) {\n    event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit >= args[\"critical_count\"])
{\n        event.check.output = \"CRITICAL: \" + args[\"critical_count\"].toString()
+ \" or more selected events are in a critical state (\" + crit.toString() +
```

```
\")\n\";\n        event.check.status = 2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn = sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n        event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected events are in a warning
state (\" + warn.toString() + \")\n\";\n        event.check.status = 1;\n
return event;\n    }\n}\n\nevent.check.output = \"Everything looks good (\" +
percentOK.toString() + \"% OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}
```

The configuration for a service component that references the `aggregate` rule template might look like this example:

YML

```
---
type: ServiceComponent
api_version: bsm/v1
metadata:
  name: webservers
spec:
  services:
    - website-services
  interval: 60
  query:
    - type: fieldSelector
      value: webserver in event.check.subscriptions
  rules:
    - template: aggregate
      name: webservers_50-70
      arguments:
        critical_threshold: 70
        warning_threshold: 50
  handlers:
    - slack
```

JSON

```
{
  "type": "ServiceComponent",
  "api_version": "bsm/v1",
```

```
  "metadata": {
    "name": "webservers"
  },
  "spec": {
    "services": [
      "website-services"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "template": "aggregate",
        "name": "webservers_50-70",
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        }
      }
    ],
    "handlers": [
      "slack"
    ]
  }
}
```

Apply rule templates to service components

Rule templates are general, parameterized resources that can apply to one or more service components. To apply a rule template to a specific service component:

- List the rule template name in the service component's `rules.template` field.
- Specify the arguments the rule template requires in the service component's `rules.template.arguments` object.

Several service components can use the same rule template with different argument values. For example, a rule template might evaluate one argument, `threshold_ok`, against the number of events with OK status, as represented by the following logic:

```
if numberEventsOK < threshold_ok {
  emit warning event
}
```

You can specify a variety of thresholds as arguments in service component definitions that reference this rule template. One service component might set a `threshold_ok` value at 10; another service component might set the value at 50. Both service components can make use of the same rule template at the threshold that makes sense for that component.

Service components can reference more than one rule template. Sensu evaluates each rule separately, and each rule produces its own event as output.

Rule template specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For rule template configuration in this version of Sensu, the api_version should always be `bsm/v1`.
required	Required for rule template configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `api_version: bsm/v1` **JSON** `{`

```
  "api_version": "bsm/v1"
}
```

metadata	
description	Top-level collection of information about the rule template, including `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`.
required	true
type	Map of key-value pairs
example	**YML** `metadata:` `  name: aggregate` `  namespace: default` `  created_by: admin` `  labels:` `  region: us-west-1` `  annotations:` `  managed_by: ops` **JSON** `{` `  "metadata": {` `  "name": "aggregate",` `  "namespace": "default",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1"` `  },` `  "annotations": {` `  "managed_by": "ops"` `  }` `  }` `}`

spec	
description	Top-level map that includes the rule template configuration spec attributes.
required	Required for rule template configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs
example	YML spec: arguments: properties: critical_count: description: create an event with a critical status if there the number of critical events is equal to or greater than this count type: number critical_threshold: description: create an event with a critical status if the percentage of non-zero events is equal to or greater than this threshold type: number metric_handlers: default: {} description: metric handlers to use for produced metrics items: type: string type: array produce_metrics: default: {} description: produce metrics from aggregate data and include them in the produced event type: boolean set_metric_annotations:

```
        default: {}
        description: annotate the produced event with
metric annotations
        type: boolean
      warning_count:
        description: create an event with a warning status
if there the number of
          critical events is equal to or greater than this
count
        type: number
      warning_threshold:
        description: create an event with a warning status
if the percentage of non-zero
          events is equal to or greater than this threshold
        type: number
    required: null
  description: Monitor a distributed service - aggregate
one or more events into a
    single event. This BSM rule template allows you to
treat the results of multiple
    disparate check executions - executed across multiple
disparate systems - as a
    single event. This template is extremely useful in
dynamic environments and/or
    environments that have a reasonable tolerance for
failure. Use this template when
    a service can be considered healthy as long as a
minimum threshold is satisfied
    (e.g. at least 5 healthy web servers? at least 70% of N
processes healthy?).
  eval: |2
    if (events && events.length == 0) {
        event.check.output = "WARNING: No events selected
for aggregate
    ";
        event.check.status = 1;
        return event;
    }
    event.annotations["io.sensu.bsm.selected_event_count"]
= events.length;
    percentOK = sensu.PercentageBySeverity("ok");
    if (!!args["produce_metrics"]) {
```

```
        var ts = Math.floor(new Date().getTime() / 1000);
        event.timestamp = ts;
        var tags = [
            {
                name: "service",
                value: event.entity.name
            },
            {
                name: "entity",
                value: event.entity.name
            },
            {
                name: "check",
                value: event.check.name
            }
        ];
        event.metrics = sensu.NewMetrics({
            points: [
                {
                    name: "percent_non_zero",
                    timestamp: ts,
                    value: sensu.PercentageBySeverity("non-
zero"),
                    tags: tags
                },
                {
                    name: "percent_ok",
                    timestamp: ts,
                    value: percentOK,
                    tags: tags
                },
                {
                    name: "percent_warning",
                    timestamp: ts,
                    value:
sensu.PercentageBySeverity("warning"),
                    tags: tags
                },
                {
                    name: "percent_critical",
                    timestamp: ts,
                    value:
```

```
sensu.PercentageBySeverity("critical"),
                    tags: tags
                },
                {
                    name: "percent_unknown",
                    timestamp: ts,
                    value:
sensu.PercentageBySeverity("unknown"),
                    tags: tags
                },
                {
                    name: "count_non_zero",
                    timestamp: ts,
                    value: sensu.CountBySeverity("non-
zero"),
                    tags: tags
                },
                {
                    name: "count_ok",
                    timestamp: ts,
                    value: sensu.CountBySeverity("ok"),
                    tags: tags
                },
                {
                    name: "count_warning",
                    timestamp: ts,
                    value:
sensu.CountBySeverity("warning"),
                    tags: tags
                },
                {
                    name: "count_critical",
                    timestamp: ts,
                    value:
sensu.CountBySeverity("critical"),
                    tags: tags
                },
                {
                    name: "count_unknown",
                    timestamp: ts,
                    value:
sensu.CountBySeverity("unknown"),
```

```
                    tags: tags
                }
            ]
        });
        if (!!args["metric_handlers"]) {
            event.metrics.handlers =
args["metric_handlers"].slice();
        }
        if (!!args["set_metric_annotations"]) {
            var i = 0;
            while(i \u003c event.metrics.points.length) {

event.annotations["io.sensu.bsm.selected_event_" +
event.metrics.points[i].name] =
event.metrics.points[i].value.toString();
                i++;
            }
        }
    }
    if (!!args["critical_threshold"] && percentOK \u003c=
args["critical_threshold"]) {
        event.check.output = "CRITICAL: Less than " +
args["critical_threshold"].toString() + "% of selected
events are OK (" + percentOK.toString() + "%)
    ";
        event.check.status = 2;
        return event;
    }
    if (!!args["warning_threshold"] && percentOK \u003c=
args["warning_threshold"]) {
        event.check.output = "WARNING: Less than " +
args["warning_threshold"].toString() + "% of selected
events are OK (" + percentOK.toString() + "%)
    ";
        event.check.status = 1;
        return event;
    }
    if (!!args["critical_count"]) {
        crit = sensu.CountBySeverity("critical");
        if (crit \u003e= args["critical_count"]) {
            event.check.output = "CRITICAL: " +
args["critical_count"].toString() + " or more selected
```

```
events are in a critical state (" + crit.toString() + ")
    ";
            event.check.status = 2;
            return event;
        }
    }
    if (!!args["warning_count"]) {
        warn = sensu.CountBySeverity("warning");
        if (warn \u003e= args["warning_count"]) {
            event.check.output = "WARNING: " +
args["warning_count"].toString() + " or more selected
events are in a warning state (" + warn.toString() + ")
    ";
            event.check.status = 1;
            return event;
        }
    }
    event.check.output = "Everything looks good (" +
percentOK.toString() + "% OK)";
    event.check.status = 0;
    return event;
```

JSON

```
{
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical
status if there the number of critical events is equal to
or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a critical
status if the percentage of non-zero events is equal to or
greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
```

```
          "description": "metric handlers to use for
produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from aggregate
data and include them in the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event with
metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a warning
status if there the number of critical events is equal to
or greater than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event with a warning
status if the percentage of non-zero events is equal to or
greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service -
aggregate one or more events into a single event. This BSM
rule template allows you to treat the results of multiple
disparate check executions - executed across multiple
disparate systems - as a single event. This template is
extremely useful in dynamic environments and/or
environments that have a reasonable tolerance for failure.
Use this template when a service can be considered healthy
```

```
as long as a minimum threshold is satisfied (e.g. at least 5
healthy web servers? at least 70% of N processes
healthy?).",
    "eval": "\nif (events \\u0026\\u0026 events.length ==
0) {\n    event.check.output = \"WARNING: No events
selected for aggregate\n\";\n    event.check.status = 1;\n
return
event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_even
t_count\"] = events.length;\n\npercentOK =
sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n    event.timestamp = ts;\n\n
var tags = [\n        {\n            name: \"service\",\n
value: event.entity.name\n        },\n        {\n
name: \"entity\",\n            value: event.entity.name\n
},\n        {\n            name: \"check\",\n
value: event.check.name\n        }\n    ];\n\n
event.metrics = sensu.NewMetrics({\n        points: [\n
{\n                name: \"percent_non_zero\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"percent_ok\",\n                timestamp: ts,\n
value: percentOK,\n                tags: tags\n
},\n            {\n                name:
\"percent_warning\",\n                timestamp: ts,\n
value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"percent_critical\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"percent_unknown\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n
name: \"count_non_zero\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"count_ok\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"ok\"),\n
tags: tags\n            },\n            {\n
```

```
name: \"count_warning\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"count_critical\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"count_unknown\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n            }\n        ]\n    });\n\n    if
(!!args[\"metric_handlers\"]) {\n
event.metrics.handlers =
args[\"metric_handlers\"].slice();\n    }\n\n    if
(!!args[\"set_metric_annotations\"]) {\n        var i =
0;\n\n        while(i \\u003c event.metrics.points.length)
{\n
event.annotations[\"io.sensu.bsm.selected_event_\" +
event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n
i++;\n        }\n    }\n}\n\nif
(!!args[\"critical_threshold\"] \\u0026\\u0026 percentOK
\\u003c= args[\"critical_threshold\"]) {\n
event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected
events are OK (\" + percentOK.toString() + \"%)\n\";\n
event.check.status = 2;\n    return event;\n}\n\nif
(!!args[\"warning_threshold\"] \\u0026\\u0026 percentOK
\\u003c= args[\"warning_threshold\"]) {\n
event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected
events are OK (\" + percentOK.toString() + \"%)\n\";\n
event.check.status = 1;\n    return event;\n}\n\nif
(!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit
\\u003e= args[\"critical_count\"]) {\n
event.check.output = \"CRITICAL: \" +
args[\"critical_count\"].toString() + \" or more selected
events are in a critical state (\" + crit.toString() +
\")\n\";\n        event.check.status = 2;\n        return
event;\n    }\n}\n\nif (!!args[\"warning_count\"]) {\n
warn = sensu.CountBySeverity(\"warning\");\n\n    if (warn
\\u003e= args[\"warning_count\"]) {\n
event.check.output = \"WARNING: \" +
```

```
args[\"warning_count\"].toString() + \" or more selected
events are in a warning state (\" + warn.toString() +
\")\n\";\n        event.check.status = 1;\n        return
event;\n    }\n}\n\nevent.check.output = \"Everything looks
good (\" + percentOK.toString() + \"%
OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}
```

type	
description	Top-level attribute that specifies the resource type. For rule template configuration, the type should always be `RuleTemplate`.
required	Required for rule template configuration in `wrapped-json` or `yaml` format.
type	String

YML

example

```
type: RuleTemplate
```

JSON

```
{
  "type": "RuleTemplate"
}
```

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's

	meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` **JSON** `{` `  "annotations": {` `  "managed-by": "ops"` `  }` `}`

created_by

description	Username of the Sensu user who created the rule template or last updated the rule template. Sensu automatically populates the `created_by` field when the rule template is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{`

```
  "created_by": "admin"
}
```

labels	
description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  region: us-west-1` **JSON** `{` `  "labels": {` `  "region": "us-west-1"` `  }` `}`

name

description	Name for the rule template that is used internally by Sensu.
required	true
type	String
example	**YML** `name: aggregate` **JSON** `{ "name": "aggregate" }`

namespace

description	Sensu RBAC namespace that the rule template belongs to.
required	true
type	String
example	**YML** `namespace: default` **JSON** `{ "namespace": "default" }`

Spec attributes

arguments	
description	The rule template's arguments using JSON Schema properties.
required	true
type	Map of key-value pairs
example	YML

```
arguments:
  properties:
    critical_count:
      description: create an event with a critical status
if there the number of critical
        events is equal to or greater than this count
      type: number
    critical_threshold:
      description: create an event with a critical status
if the percentage of non-zero
        events is equal to or greater than this threshold
      type: number
    metric_handlers:
      default: {}
      description: metric handlers to use for produced
metrics
      items:
        type: string
      type: array
    produce_metrics:
      default: {}
      description: produce metrics from aggregate data and
include them in the produced
        event
      type: boolean
    set_metric_annotations:
      default: {}
      description: annotate the produced event with metric
annotations
      type: boolean
```

```
    warning_count:
      description: create an event with a warning status if
there the number of critical
        events is equal to or greater than this count
      type: number
    warning_threshold:
      description: create an event with a warning status if
the percentage of non-zero
        events is equal to or greater than this threshold
      type: number
  required: null
```

JSON

```
{
  "arguments": {
    "properties": {
      "critical_count": {
        "description": "create an event with a critical
status if there the number of critical events is equal to
or greater than this count",
        "type": "number"
      },
      "critical_threshold": {
        "description": "create an event with a critical
status if the percentage of non-zero events is equal to or
greater than this threshold",
        "type": "number"
      },
      "metric_handlers": {
        "default": {
        },
        "description": "metric handlers to use for produced
metrics",
        "items": {
          "type": "string"
        },
        "type": "array"
      },
      "produce_metrics": {
        "default": {
        },
```

```
        "description": "produce metrics from aggregate data
and include them in the produced event",
        "type": "boolean"
      },
      "set_metric_annotations": {
        "default": {
        },
        "description": "annotate the produced event with
metric annotations",
        "type": "boolean"
      },
      "warning_count": {
        "description": "create an event with a warning
status if there the number of critical events is equal to
or greater than this count",
        "type": "number"
      },
      "warning_threshold": {
        "description": "create an event with a warning
status if the percentage of non-zero events is equal to or
greater than this threshold",
        "type": "number"
      }
    },
    "required": null
  }
}
```


description	
description	Plain text description of the rule template's behavior.
required	true
type	String YML
example	`description: Monitor a distributed service - aggregate one or more events into a single event. This BSM rule template`

```
allows you to treat the results of multiple disparate check executions - executed across multiple disparate systems - as a single event. This template is extremely useful in dynamic environments and/or environments that have a reasonable tolerance for failure. Use this template when a service can be considered healthy as long as a minimum threshold is satisfied (e.g. at least 5 healthy web servers? at least 70% of N processes healthy?).
```

JSON

```
{
  "description": "Monitor a distributed service - aggregate one or more events into a single event. This BSM rule template allows you to treat the results of multiple disparate check executions - executed across multiple disparate systems - as a single event. This template is extremely useful in dynamic environments and/or environments that have a reasonable tolerance for failure. Use this template when a service can be considered healthy as long as a minimum threshold is satisfied (e.g. at least 5 healthy web servers? at least 70% of N processes healthy?)."
}
```

eval	
description	ECMAScript 5 (JavaScript) expression for the rule template to evaluate.
required	true
type	String
example	**YML** `eval: |2` `  if (events && events.length == 0) {` `  event.check.output = "WARNING: No events selected for aggregate`

```
  ";
      event.check.status = 1;
      return event;
  }
  event.annotations["io.sensu.bsm.selected_event_count"] =
events.length;
  percentOK = sensu.PercentageBySeverity("ok");
  if (!!args["produce_metrics"]) {
      var ts = Math.floor(new Date().getTime() / 1000);
      event.timestamp = ts;
      var tags = [
          {
              name: "service",
              value: event.entity.name
          },
          {
              name: "entity",
              value: event.entity.name
          },
          {
              name: "check",
              value: event.check.name
          }
      ];
      event.metrics = sensu.NewMetrics({
          points: [
              {
                  name: "percent_non_zero",
                  timestamp: ts,
                  value: sensu.PercentageBySeverity("non-
zero"),
                  tags: tags
              },
              {
                  name: "percent_ok",
                  timestamp: ts,
                  value: percentOK,
                  tags: tags
              },
              {
                  name: "percent_warning",
                  timestamp: ts,
```

```
                    value:
sensu.PercentageBySeverity("warning"),
                    tags: tags
                },
                {
                    name: "percent_critical",
                    timestamp: ts,
                    value:
sensu.PercentageBySeverity("critical"),
                    tags: tags
                },
                {
                    name: "percent_unknown",
                    timestamp: ts,
                    value:
sensu.PercentageBySeverity("unknown"),
                    tags: tags
                },
                {
                    name: "count_non_zero",
                    timestamp: ts,
                    value: sensu.CountBySeverity("non-zero"),
                    tags: tags
                },
                {
                    name: "count_ok",
                    timestamp: ts,
                    value: sensu.CountBySeverity("ok"),
                    tags: tags
                },
                {
                    name: "count_warning",
                    timestamp: ts,
                    value: sensu.CountBySeverity("warning"),
                    tags: tags
                },
                {
                    name: "count_critical",
                    timestamp: ts,
                    value: sensu.CountBySeverity("critical"),
                    tags: tags
                },
```

```
                {
                    name: "count_unknown",
                    timestamp: ts,
                    value: sensu.CountBySeverity("unknown"),
                    tags: tags
                }
            ]
        });
        if (!!args["metric_handlers"]) {
            event.metrics.handlers =
args["metric_handlers"].slice();
        }
        if (!!args["set_metric_annotations"]) {
            var i = 0;
            while(i \u003c event.metrics.points.length) {

event.annotations["io.sensu.bsm.selected_event_" +
event.metrics.points[i].name] =
event.metrics.points[i].value.toString();
                i++;
            }
        }
    }
    if (!!args["critical_threshold"] && percentOK \u003c=
args["critical_threshold"]) {
        event.check.output = "CRITICAL: Less than " +
args["critical_threshold"].toString() + "% of selected
events are OK (" + percentOK.toString() + "%)
    ";
        event.check.status = 2;
        return event;
    }
    if (!!args["warning_threshold"] && percentOK \u003c=
args["warning_threshold"]) {
        event.check.output = "WARNING: Less than " +
args["warning_threshold"].toString() + "% of selected
events are OK (" + percentOK.toString() + "%)
    ";
        event.check.status = 1;
        return event;
    }
    if (!!args["critical_count"]) {
```

```
      crit = sensu.CountBySeverity("critical");
      if (crit \u003e= args["critical_count"]) {
          event.check.output = "CRITICAL: " +
args["critical_count"].toString() + " or more selected
events are in a critical state (" + crit.toString() + ")
  ";
          event.check.status = 2;
          return event;
      }
  }
  if (!!args["warning_count"]) {
      warn = sensu.CountBySeverity("warning");
      if (warn \u003e= args["warning_count"]) {
          event.check.output = "WARNING: " +
args["warning_count"].toString() + " or more selected
events are in a warning state (" + warn.toString() + ")
  ";
          event.check.status = 1;
          return event;
      }
  }
  event.check.output = "Everything looks good (" +
percentOK.toString() + "% OK)";
  event.check.status = 0;
  return event;
```

JSON

```
{
  "eval": "  if (events \\u0026\\u0026 events.length == 0)
{\n      event.check.output = \"WARNING: No events selected
for aggregate\n  \";\n      event.check.status = 1;\n
return event;\n  }\n
event.annotations[\"io.sensu.bsm.selected_event_count\"] =
events.length;\n  percentOK =
sensu.PercentageBySeverity(\"ok\");\n  if
(!!args[\"produce_metrics\"]) {\n      var ts =
Math.floor(new Date().getTime() / 1000);\n
event.timestamp = ts;\n      var tags = [\n          {\n
name: \"service\",\n              value:
event.entity.name\n          },\n          {\n
name: \"entity\",\n              value: event.entity.name\n
```

```
},\n          {\n              name: \"check\",\n
value: event.check.name\n          }\n      ];\n
event.metrics = sensu.NewMetrics({\n          points: [\n
{\n                  name: \"percent_non_zero\",\n
timestamp: ts,\n                  value:
sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n              },\n              {\n
name: \"percent_ok\",\n                  timestamp: ts,\n
value: percentOK,\n                  tags: tags\n
},\n              {\n                  name:
\"percent_warning\",\n                  timestamp: ts,\n
value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n              },\n              {\n
name: \"percent_critical\",\n                  timestamp:
ts,\n                  value:
sensu.PercentageBySeverity(\"critical\"),\n
tags: tags\n              },\n              {\n
name: \"percent_unknown\",\n                  timestamp:
ts,\n                  value:
sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n              },\n              {\n
name: \"count_non_zero\",\n                  timestamp:
ts,\n                  value: sensu.CountBySeverity(\"non-
zero\"),\n                  tags: tags\n              },\n
{\n                  name: \"count_ok\",\n
timestamp: ts,\n                  value:
sensu.CountBySeverity(\"ok\"),\n                  tags:
tags\n              },\n              {\n
name: \"count_warning\",\n                  timestamp:
ts,\n                  value:
sensu.CountBySeverity(\"warning\"),\n
tags: tags\n              },\n              {\n
name: \"count_critical\",\n                  timestamp:
ts,\n                  value:
sensu.CountBySeverity(\"critical\"),\n
tags: tags\n              },\n              {\n
name: \"count_unknown\",\n                  timestamp:
ts,\n                  value:
sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n              }\n          ]\n      });\n
if (!!args[\"metric_handlers\"]) {\n
event.metrics.handlers =
```

```
args[\"metric_handlers\"].slice();\n      }\n      if
(!!args[\"set_metric_annotations\"]) {\n          var i =
0;\n          while(i \\u003c event.metrics.points.length)
{\n
event.annotations[\"io.sensu.bsm.selected_event_\" +
event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n
i++;\n          }\n      }\n  }\n  if
(!!args[\"critical_threshold\"] \\u0026\\u0026 percentOK
\\u003c= args[\"critical_threshold\"]) {\n
event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected
events are OK (\" + percentOK.toString() + \"%)\n  \";\n
event.check.status = 2;\n      return event;\n  }\n  if
(!!args[\"warning_threshold\"] \\u0026\\u0026 percentOK
\\u003c= args[\"warning_threshold\"]) {\n
event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected
events are OK (\" + percentOK.toString() + \"%)\n  \";\n
event.check.status = 1;\n      return event;\n  }\n  if
(!!args[\"critical_count\"]) {\n      crit =
sensu.CountBySeverity(\"critical\");\n      if (crit
\\u003e= args[\"critical_count\"]) {\n
event.check.output = \"CRITICAL: \" +
args[\"critical_count\"].toString() + \" or more selected
events are in a critical state (\" + crit.toString() +
\")\n  \";\n          event.check.status = 2;\n
return event;\n      }\n  }\n  if
(!!args[\"warning_count\"]) {\n      warn =
sensu.CountBySeverity(\"warning\");\n      if (warn
\\u003e= args[\"warning_count\"]) {\n
event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected
events are in a warning state (\" + warn.toString() + \")\n
\";\n          event.check.status = 1;\n          return
event;\n      }\n  }\n  event.check.output = \"Everything
looks good (\" + percentOK.toString() + \"% OK)\";\n
event.check.status = 0;\n  return event;"
}
```

Arguments attributes

properties	
description	List of properties that define the argument's behavior. In JSON Schema.
required	true
type	Array
example	YML

```yaml
properties:
  critical_count:
    description: create an event with a critical status if
there the number of
      critical events is equal to or greater than this
count
    type: number
  critical_threshold:
    description: create an event with a critical status if
the percentage of non-zero
      events is equal to or greater than this threshold
    type: number
  metric_handlers:
    default: {}
    description: metric handlers to use for produced
metrics
    items:
      type: string
    type: array
  produce_metrics:
    default: {}
    description: produce metrics from aggregate data and
include them in the produced
      event
    type: boolean
  set_metric_annotations:
    default: {}
    description: annotate the produced event with metric
annotations
    type: boolean
```

```
  warning_count:
    description: create an event with a warning status if
there the number of
      critical events is equal to or greater than this
count
    type: number
  warning_threshold:
    description: create an event with a warning status if
the percentage of non-zero
      events is equal to or greater than this threshold
    type: number
```

JSON

```
{
  "properties": {
    "critical_count": {
      "description": "create an event with a critical
status if there the number of critical events is equal to
or greater than this count",
      "type": "number"
    },
    "critical_threshold": {
      "description": "create an event with a critical
status if the percentage of non-zero events is equal to or
greater than this threshold",
      "type": "number"
    },
    "metric_handlers": {
      "default": {},
      "description": "metric handlers to use for produced
metrics",
      "items": {
        "type": "string"
      },
      "type": "array"
    },
    "produce_metrics": {
      "default": {},
      "description": "produce metrics from aggregate data
and include them in the produced event",
```

```
      "type": "boolean"
    },
    "set_metric_annotations": {
      "default": {},
      "description": "annotate the produced event with
metric annotations",
      "type": "boolean"
    },
    "warning_count": {
      "description": "create an event with a warning status
if there the number of critical events is equal to or
greater than this count",
      "type": "number"
    },
    "warning_threshold": {
      "description": "create an event with a warning status
if the percentage of non-zero events is equal to or greater
than this threshold",
      "type": "number"
    }
  },
  "required": null
}
```

required	
description	List of attributes the rule template argument requires. The listed attributes must be configured in the properties object.
required	false
type	Array

YML

example

```
required: null
```

JSON

```
```

```
{
  "required": null
}
```

Service components reference

***COMMERCIAL FEATURE**: Access business service monitoring (BSM), including service components, in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

Service components are resources for defining and managing elements of a business service in business service monitoring. A service entity consists of a number of underlying service components. A service component is a meaningful selection of Sensu events for a business service, such as database monitoring events.

A service component includes event selectors to define the events that the component includes, a service component scheduler (either an interval or cron expression), and references to at least one monitoring rule template with arguments. The monitoring rules are evaluated against aggregate data derived from the service component's selection of events. Monitoring rules are configured in a separate resource: rule templates.

If you delete a resource (for example, an entity, check, or event) that is part of one or more service components, Sensu will automatically remove the deleted resource from the service components.

Service component example

The example service component below is a dependency of the business service entity `website-services`. Sensu will execute the component at 60-second intervals for `website-services` service entities whose events include the `webserver` subscription. The monitoring rule template for the service component is `aggregate`.

YML

```
---
type: ServiceComponent
api_version: bsm/v1
metadata:
```

```
  name: webservers
spec:
  handlers:
  - slack
  interval: 60
  query:
  - type: fieldSelector
    value: webserver in event.check.subscriptions
  rules:
  - arguments:
      critical_threshold: 70
      warning_threshold: 50
    name: webservers_50-70
    template: aggregate
  services:
  - website-services
```

JSON

```
{
  "type": "ServiceComponent",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "webservers"
  },
  "spec": {
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        },
```

```
        "name": "webservers_50-70",
        "template": "aggregate"
      }
    ],
    "services": [
      "website-services"
    ]
  }
}
```

Service component scheduling

Sensu executes service components on sensu-backend processes in a round-robin fashion and according to a schedule specified by an interval or a cron expression in the component definition.During each execution of the service component, Sensu retrieves the events identified in the component's query expression and processes these events according to the monitoring rules specified in the service component definition.The rules can emit new events based on the component input.

Service component specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For service component configuration in this version of Sensu, the api_version should always be `bsm/v1`.
required	Required for service component configuration in `wrapped-json` or `yaml` format.
type	String
example	YAML `api_version: bsm/v1`

JSON

```
{
  "api_version": "bsm/v1"
}
```

metadata

description	Top-level collection of information about the service component, including `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`.
required	true
type	Map of key-value pairs

YML

example

```
metadata:
  name: webservers
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    managed_by: ops
```

JSON

```
{
  "metadata": {
    "name": "webservers",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "managed_by": "ops"
```

```
    }
  }
}
```

spec	
description	Top-level map that includes the service component configuration spec attributes.
required	Required for service component configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs
example	**YML** `spec:` `  handlers:` `  - slack` `  interval: 60` `  query:` `  - type: fieldSelector` `  value: webserver in event.check.subscriptions` `  rules:` `  - arguments:` `  critical_threshold: 70` `  warning_threshold: 50` `  name: webservers_50-70` `  template: aggregate` `  services:` `  - website-services` **JSON** `{` `  "spec": {` `  "handlers": [` `  "slack"` `  ],`

```
      "interval": 60,
      "query": [
        {
          "type": "fieldSelector",
          "value": "webserver in event.check.subscriptions"
        }
      ],
      "rules": [
        {
          "arguments": {
            "critical_threshold": 70,
            "warning_threshold": 50
          },
          "name": "webservers_50-70",
          "template": "aggregate"
        }
      ],
      "services": [
        "website-services"
      ]
    }
}
```

type	
description	Top-level attribute that specifies the resource type. For service component configuration, the type should always be `ServiceComponent`.
required	Required for service component configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `type: ServiceComponent` **JSON**

```
{
  "type": "ServiceComponent"
}
```

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` **JSON** `{` `  "annotations": {` `  "managed-by": "ops"` `  }` `}`

created_by

description	Username of the Sensu user who created or last updated the service component. Sensu automatically populates the `created_by` field when the service component is created or updated.
required	false
type	String
example	**YML** ```yaml created_by: admin ``` **JSON** ```json { "created_by": "admin" } ```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example	**YML** ```yaml labels: region: us-west-1 ``` **JSON** ```json { "labels": { "region": "us-west-1" } } ```

name

description	Name for the service component that is used internally by Sensu.
required	true
type	String
example	**YML** ```yaml name: webservers ``` **JSON** ```json { "name": "webservers" } ```

namespace

description	Sensu RBAC namespace that the service component belongs to.

required	true
type	String
example	**YML** `namespace: default` **JSON** `{ "namespace": "default" }`

Spec attributes

cron

description	When the service component should be executed, using cron syntax or a predefined schedule. Use a prefix of `TZ=` or `CRON_TZ=` to set a timezone for the cron attribute. ***NOTE**: If you're using YAML to create a service component that uses cron scheduling and the first character of the cron schedule is an asterisk (`*`), place the entire cron schedule inside single or double quotes (for example, `cron: '* * * * *'`).*
required	true (unless `interval` is configured)
type	String
example	**YML** `cron: 0 0 * * *` **JSON**

```
{
  "cron": "0 0 * * *"
}
```

handlers

description	List of handlers to use for the events the service component produces. The service component will set the handlers property in events that are produced by rule evaluation. If no handlers are specified in the service component definition, handlers can be set by the monitoring rule itself via template arguments. Handlers specified in the service component definition will override any handlers set by rule evaluation.
required	false
type	Array

example

YML

```
handlers:
  - slack
```

JSON

```
{
  "handlers": [
    "slack"
  ]
}
```

interval

description	How often the service component should be executed. In seconds. Each service component must have a value for either the `interval` or the `cron` attribute, but not both.

required	true (unless `cron` is configured)
type	Integer

YML

example

```
interval: 60
```

JSON

```
{
  "interval": 60
}
```

query

description	Query expression that describes the events that each monitoring rule should process for the service component. Read query attributes for details.
required	true
type	Array

YML

example

```
query:
  - type: fieldSelector
    value: webserver in event.check.subscriptions
```

JSON

```
{
  "query": [
    {
      "type": "fieldSelector",
      "value": "webserver in event.check.subscriptions"
    }
  ]
}
```

rules

description	List of the rule templates and arguments that Sensu should apply for the service component. Sensu evaluates each rule separately, and each rule produces its own event as output. Read rules attributes for details.
required	true
type	Map of key-value pairs

example

YML

```
rules:
- arguments:
    critical_threshold: 70
    warning_threshold: 50
  name: webservers_50-70
  template: aggregate
```

JSON

```
{
  "rules": [
    {
      "arguments": {
        "critical_threshold": 70,
        "warning_threshold": 50
      },
      "name": "webservers_50-70",
      "template": "aggregate"
    }
  ]
}
```

services

description	List of business service entities that include the service component as a dependency.
required	true
type	Array
example	**YML** `services:` `  - website-services` **JSON** `{` `  "services": [` `  "website-services"` `  ]` `}`

Query attributes

type	
description	Type of selector to use to identify the events that the service component's monitoring rule should process: `fieldSelector` or `labelSelector`.
required	true
type	String
example	**YML** `type: fieldSelector` **JSON** `{`

```
  "type": "fieldSelector"
}
```

value	
description	Selector expression the query will use to identify the events that the service component's monitoring rule should process.
required	true
type	String

YML

```
value: webserver in event.check.subscriptions
```

JSON

```
{
  "value": "webserver in event.check.subscriptions"
}
```

Rules attributes

arguments	
description	The arguments to pass to the rule template for the service component. Argument names and values will vary depending on the arguments configured in the specified rule template.
required	false
type	Map of key-value pairs

YML

```
```

```
- arguments:
    critical_threshold: 70
    warning_threshold: 50
```

JSON

```
{
  "arguments": {
    "critical_threshold": 70,
    "warning_threshold": 50
  }
}
```

name

description	Explicit name to use for the rule-specific events generated for the service component. These names help keep events distinct when a service component includes different rules for the same rule template.
required	true
type	String

YML

example

```
name: webservers_50-70
```

JSON

```
{
  "name": "webservers_50-70"
}
```

template

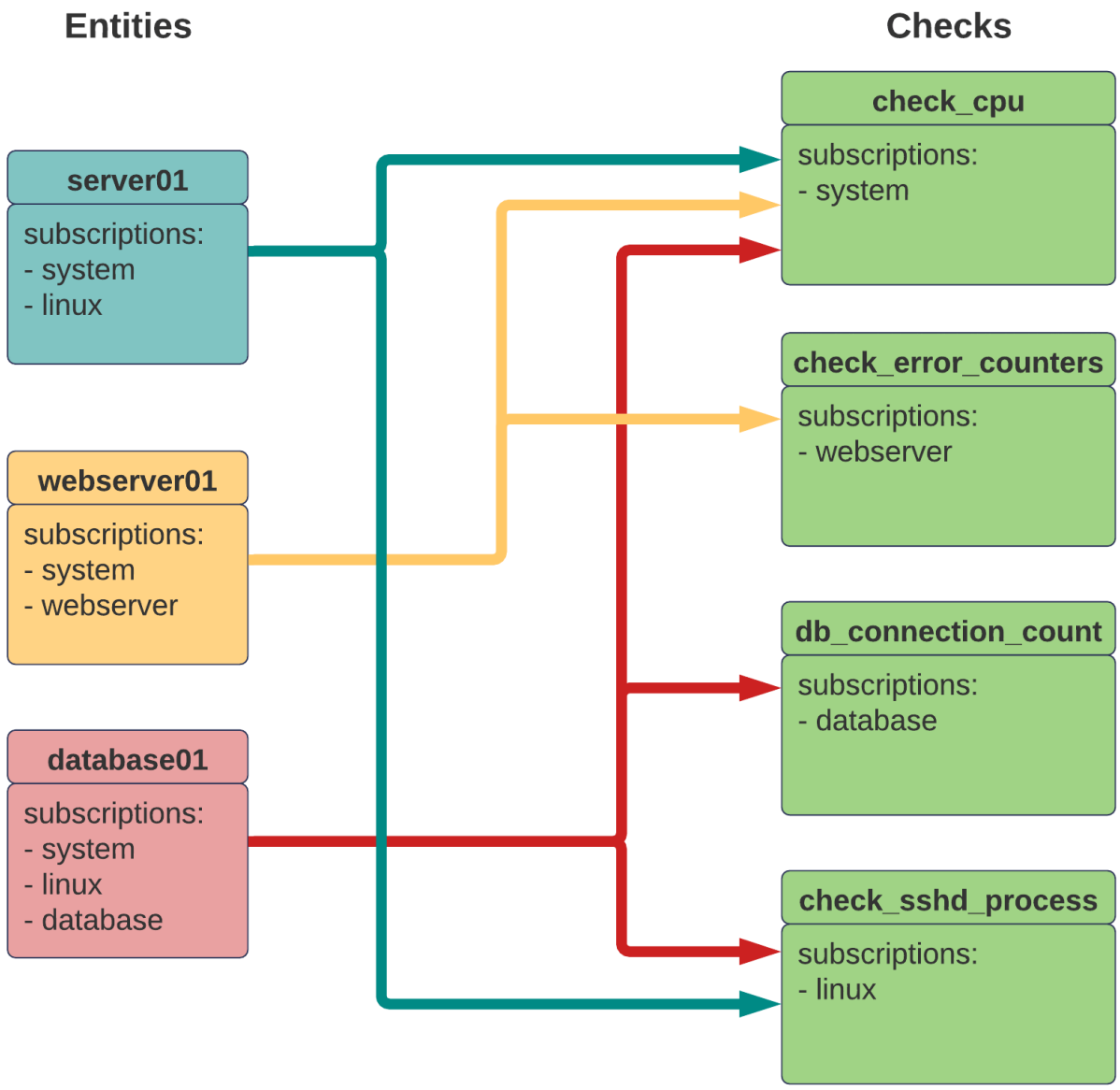
description	Name of the rule template the service component should use.
required	true
type	String
example	**YML** `template: aggregate` **JSON** `{ "template": "aggregate" }`

Subscriptions reference

Sensu uses the publish/subscribe model of communication. The publish/subscribe model is powerful in ephemeral or elastic infrastructures, where the names and numbers of things change over time.

Because Sensu uses the publish/subscribe model, you can write checks even if you don't know the specific names of the entities that should run the checks. Likewise, your entities do not need to know the specific names of the checks they should execute. The Sensu backend coordinates check execution for you by comparing the subscriptions you specify in your checks and entities to determine which entities should receive execution requests for a given check.

The diagram below shows how Sensu coordinates check execution based on subscriptions. For example, the `check_cpu` check includes the `system` subscription. All three entities include the `system` subscription, so all three entities will execute the `check_cpu` check. However, only the `server01` and `database01` entities will execute `check_sshd_process` — the `webserver01` entity does not include the `linux` subscription required to execute `check_sshd_process`.



Sensu subscriptions are equivalent to topics in a traditional publish/subscribe system. Sensu entities become subscribers to these topics via the strings you specify with the agent `subscriptions` flag. Sensu checks have a `subscriptions` attribute, where you specify strings to indicate which subscribers will execute the checks. For Sensu to execute a check, the check definition must include a subscription that matches the subscription of at least one Sensu entity.

***NOTE**: Proxy entities do not use subscriptions. Instead, use proxy checks to generate events for proxy entities.*

As loosely coupled references, subscriptions avoid the fragility of traditional host-based monitoring systems. Subscriptions allow you to configure check requests in a one-to-many model for entire groups or subgroups of entities rather than a traditional one-to-one mapping of configured hosts or

observability checks.

Subscription example

Suppose you have a Sensu agent entity with the `linux` subscription:

```
sensu-agent start --subscriptions linux --log-level debug
```

For this agent to run a check, you must have at least one check with `linux` specified in the `subscriptions` attribute, such as this check to collect status information:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: collect_info
spec:
  command: collect.sh
  handlers:
  - slack
  interval: 10
  publish: true
  subscriptions:
  - linux
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "command": "collect.sh",
    "handlers": [
```

```
        "slack"
      ],
      "interval": 10,
      "publish": true,
      "subscriptions": [
        "linux"
      ]
    }
  }
```

If this is your only check for the `linux` subscription, this is the only check that your agent will execute. If you add more checks that specify the `linux` subscription, your agent will automatically run those checks too (as long as the `publish` attribute is set to `true` in the check definitions).

You can also add more subscriptions for your entity. For example, if you want your agent entity to execute checks for the `webserver` subscription, you can add it with the `subscriptions` flag:

```
sensu-agent start --subscriptions linux,webserver --log-level debug
```

Now your agent entity will execute checks with the `linux` or `webserver` subscriptions.

To directly add, update, and delete subscriptions for individual entities, use sensuctl, the core/v2/entities API endpoints, or the web UI.

Configure subscriptions

Sensu automatically executes a check when the check definition includes a subscription that matches a subscription for at least one Sensu entity. In other words, subscriptions are configured for both checks and agent entities:

- To configure subscriptions for a check, add one or more subscription names in the check `subscriptions` attribute.
- To configure subscriptions for an agent entity, specify a subscription that matches one subscription in each check that the agent's entities should execute.

The Sensu backend schedules checks once per interval for each agent entity with a matching subscription. For example, if you have three entities configured with the `system` subscription, a check

configured with the `system` subscription results in three monitoring events per interval: one check execution per entity per interval.

> ***WARNING**: Make sure that your checks and entities share only one subscription. Entities receive a separate check request for each matching subscription, even if the requests are for the same check. This can result in check execution errors as well as unexpected results for check `history` and the features that rely on it.*

In addition to the subscriptions defined in the agent configuration, Sensu agent entities subscribe automatically to subscriptions that match their entity `name`. For example, an agent entity with `name: "i-424242"` subscribes to check requests with the subscription `entity:i-424242`. This makes it possible to generate ad hoc check requests that target specific entities via the API.

> ***NOTE**: You can directly add, update, and delete subscriptions for individual entities via the backend with sensuctl, the core/v2/entities API endpoints, and the web UI.*

Publish checks

If you want Sensu to automatically schedule and execute a check according to its subscriptions, set the `publish` attribute to `true` in the check definition.

You can also manually schedule ad hoc check execution with the core/v2/checks API endpoints, whether the `publish` attribute is set to `true` or `false`. To target the subscriptions defined in the check, include only the check name in the request body (for example, `"check": "check_cpu"`). To override the check's subscriptions and target an alternate entity or group of entities, add the subscriptions attribute to the request body:

```
{
  "check": "check_cpu",
  "subscriptions": [
    "entity:i-424242",
    "entity:i-828282"
  ]
}
```

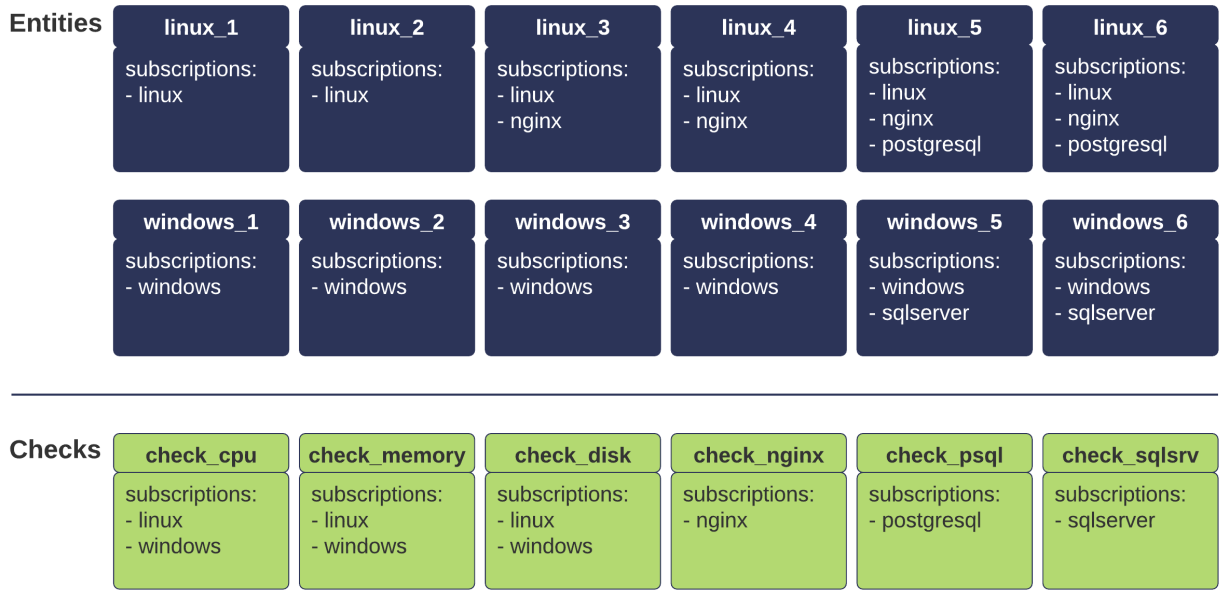
Monitor multiple servers

You can use subscriptions to configure monitoring and observability for multiple servers with different operating systems and monitoring requirements.

For example, suppose you want to set up monitoring for these servers:

- Six Linux servers:
  - Get CPU, memory, and disk status for all six
  - Get NGINX metrics for four
  - Get PostgreSQL metrics for two
- Six Windows servers:
  - Get CPU, memory, and disk checks for all six
  - Get SQL Server metrics for two

This diagram shows the subscriptions to list for each of the 12 servers (the entities) and for each check to achieve the example monitoring configuration:



In this scenario, none of the Windows servers should execute the NGINX metrics check, so the `check_nginx` subscriptions do not match any subscriptions listed for any of the Windows servers. Two of the six Windows servers *should* execute the SQL Server metrics check, so the subscription listed in the `check sqlsrv` definition matches a subscription listed for those two Windows server entities.

Subscription naming considerations

Consistent subscription naming helps you group and filter different entities and quickly understand which entities will be affected by any changes.

Subscriptions based on function are helpful when you're creating silences. For example, if you need to silence all webservers for maintenance, it's easier to silence the affected entities if they all include a subscription like `webserver` instead of identifying and silencing all of your webserver entities individually. Other function-based subscriptions might be `database`, `switch`, `service`, or `container`.

Subscription naming is also important in the context of API and sensuctl filters and web UI searches. Consistent subscription naming means that search queries like `"linux" in checks.subscriptions` will actually retrieve all of your checks that run on Linux agents.

To make subscriptions more granular, use camel case to append information about environment, roles, entity type, or any other category as needed. For example, you can use `webserverDev` and `webserverProd` to specify a distinction between development and production webservers while preserving your ability to find all webserver entities with a search query like `"webserver" in entity.subscriptions`.

Tokens reference

Tokens are placeholders in a check definition that the agent replaces with entity information before executing the check. You can use tokens to fine-tune check attributes (like alert thresholds) on a per-entity level while reusing the check definition.

When a check is scheduled to be executed by an agent, it first goes through a token substitution step. The agent replaces any tokens with matching attributes from the entity definition, and then the check is executed. Invalid templates or unmatched tokens return an error, which is logged and sent to the Sensu backend message transport. Checks with token-matching errors are not executed.

Token substitution is supported for check, hook, and dynamic runtime asset definitions. Only entity attributes are available for substitution. Token substitution is not available for event filters because filters already have access to the entity.

Available entity attributes will always have string values, such as labels and annotations.

Example: Token substitution for check thresholds

This example demonstrates a reusable disk usage check. The check command includes `-w` (warning) and `-c` (critical) arguments with default values for the thresholds (as percentages) for generating warning or critical events. The check will compare every subscribed entity's disk space against the default threshold values to determine whether to generate a warning or critical event.

However, the check command also includes token substitution, which means you can add entity labels that correspond to the check command tokens to specify different warning and critical values for individual entities. Instead of creating a different check for every set of thresholds, you can use the same check to apply the defaults in most cases and the token-substituted values for specific entities.

Follow this example to set up a reusable check for disk usage:

1. Add the sensu/check-disk-usage dynamic runtime asset, which includes the command you will need for your check:

```
sensuctl asset add sensu/check-disk-usage:0.6.0
```

You will receive a response to confirm that the asset was added:

```
fetching bonsai asset: sensu/check-disk-usage:0.6.0
added asset: sensu/check-disk-usage:0.6.0

You have successfully added the Sensu asset resource, but the asset will not
get downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset
to the appropriate
resource, populate the "runtime_assets" field with ["sensu/check-disk-usage].
```

2. Create the `check-disk-usage` check:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check-disk-usage
spec:
  check_hooks: []
  command: check-disk-usage -w {{index .labels "disk_warning" | default 80}} -
c
    {{.labels.disk_critical | default 90}}
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  output_metric_tags: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu/check-disk-usage
  stdin: false
```

```
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-disk-usage"
  },
  "spec": {
    "check_hooks": [],
    "command": "check-disk-usage -w {{index .labels "disk_warning" | default
80}} -c {{.labels.disk_critical | default 90}}",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "output_metric_tags": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu/check-disk-usage"
    ],
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
```

```
}
EOF
```

This check will run on every entity with the subscription `system`. According to the default values in the command, the check will generate a warning event at 80% disk usage and a critical event at 90% disk usage.

3. To receive alerts at different thresholds for an existing entity with the `system` subscription, add `disk_warning` and `disk_critical` labels to the entity.

   Use sensuctl to open an existing entity in a text editor:

   ```
   sensuctl edit entity ENTITY_NAME
   ```

   And add the following labels in the entity metadata:

   ```
   labels:
     disk_warning: "65"
     disk_critical: "75"
   ```

After you save your changes, the `check-disk-usage` check will substitute the `disk_warning` and `disk_critical` label values to generate events at 65% and 75% of disk usage, respectively, for this entity only. The check will continue to use the 80% and 90% default values for other subscribed entities.

Add a hook that uses token substitution

Now you have a resusable check that will send disk usage alerts at default or entity-specific thresholds. You may want to add a hook to list more details about disk usage for warning and critical events.

The hook in this example will list disk usage in human-readable format, with error messages filtered from the hook output. By default, the hook will list details for the top directory and the first layer of subdirectories. As with the `check-disk-usage` check, you can add a `disk_usage_root` label to individual entities to specify a different directory for the hook via token substitution.

1. Add the hook definition:

SHELL

```
cat << EOF | sensuctl create
---
type: HookConfig
api_version: core/v2
metadata:
  name: disk_usage_details
spec:
  command: du -h --max-depth=1 -c {{index .labels "disk_usage_root" | default
"/"}}  2>/dev/null
  runtime_assets: null
  stdin: false
  timeout: 60
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "disk_usage_details"
  },
  "spec": {
    "command": "du -h --max-depth=1 -c {{index .labels "disk_usage_root" |
default \"/\"}}  2>/dev/null",
    "runtime_assets": null,
    "stdin": false,
    "timeout": 60
  }
}
EOF
```

2. Add the hook to the `check-disk-usage` check.

   Use sensuctl to open the check in a text editor:

```
```

```
sensuctl edit check check-disk-usage
```

Update the check definition to include the `disk_usage_details` hook for `non-zero` events:

```
check_hooks:
- non-zero:
  - disk_usage_details
```

3. As with the disk usage check command, the hook command includes a token substitution option To use a specific directory instead of the default for specific entities, edit the entity definition to add a `disk_usage_root` label and specify the directory:

   Use sensuctl to open the entity in a text editor:

   ```
   sensuctl edit entity ENTITY_NAME
   ```

   Add the `disk_usage_root` label with the desired substitute directory in the entity metadata:

   ```
   labels:
     disk_usage_root: "/substitute-directory"
   ```

After you save your changes, for this entity, the hook will substitute the directory you specified for the `disk_usage_root` label to provide additional disk usage details for every non-zero event the `check-disk-usage` check generates.

Manage entity labels

You can use token substitution with any defined entity attributes, including custom labels. Read the entities reference for information about managing entity labels for proxy entities and agent entities.

Manage dynamic runtime assets

You can use token substitution in the URLs of your your dynamic runtime asset definitions. Token substitution allows you to host your dynamic runtime assets at different URLs (such as at different datacenters) without duplicating your assets, as shown in the following example:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: sensu-go-hello-world
spec:
  builds:
  - sha512:
07665fda5b7c75e15e4322820aa7ddb791cc9338e38444e976e601bc7d7970592e806a7b88733690a238
b7325437d31f85e98ae2fe47b008ca09c86530da9600
    url: "{{ .labels.asset_url }}/sensu-go-hello-world-0.0.1.tar.gz"
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-go-hello-world"
  },
  "spec": {
    "builds": [
      {
        "sha512":
"07665fda5b7c75e15e4322820aa7ddb791cc9338e38444e976e601bc7d7970592e806a7b88733690a23
8b7325437d31f85e98ae2fe47b008ca09c86530da9600",
        "url": "{{ .labels.asset_url }}/sensu-go-hello-world-0.0.1.tar.gz"
      }
    ]
  }
}
```

With this asset definition, which includes the `.labels.asset_url` token substitution, checks and hooks can include `sensu-go-hello-world` as a dynamic runtime assets and Sensu Go will use the

token substitution for the agent's entity. Handlers and mutators can also include `sensu-go-hello-world` as a dynamic runtime asset, but Sensu Go will use the token subtitution for the backend's entity instead of the agent's entity.

You can also use token substitution to customize dynamic runtime asset headers (for example, to include secure information for authentication). Sensu also provides an `assetPath` function that allows you to substitute a dynamic runtime asset's local path on disk.

> ***NOTE**: To maintain security, you cannot use token substitution for a dynamic runtime asset's SHA512 value.*

Token specification

Sensu Go uses the Go template package to implement token substitution. Use double curly braces around the token and a dot before the attribute to be substituted: `{{ .system.hostname }}`.

Token substitution syntax

Tokens are invoked by wrapping references to entity attributes and labels with double curly braces, such as `{{ .name }}` to substitute an entity's name. Access nested Sensu entity attributes with dot notation (for example, `system.arch`).

- `{{ .name }}` would be replaced with the entity `name` attribute
- `{{ .labels.url }}` would be replaced with a custom label called `url`
- `{{ .labels.disk_warning }}` would be replaced with a custom label called `disk_warning`
- `{{ index .labels "disk_warning" }}` would be replaced with a custom label called `disk_warning`
- `{{ index .labels "cpu.threshold" }}` would be replaced with a custom label called `cpu.threshold`

> ***NOTE**: When an annotation or label name has a dot (for example, `cpu.threshold`), you must use the template index function syntax to ensure correct processing because the dot notation is also used for object nesting.*

Token substitution default values

If an attribute is not provided by the entity, a token's default value will be substituted. Token default values are separated by a pipe character and the word "default" (`| default`). Use token default values to provide a fallback value for entities that are missing a specified token attribute.

For example, `{{.labels.url | default "https://sensu.io"}}` would be replaced with a custom label called `url` . If no such attribute called `url` is included in the entity definition, the default (or fallback) value of `https://sensu.io` will be used to substitute the token.

Token substitution with quoted strings

You can escape quotes to express quoted strings in token substitution templates as shown in the Go template package examples. For example, to provide `"substitution"` as a default value for entities that are missing the `website` attribute (including the quotation marks):

```
{{ .labels.website | default "\"substitution\"" }}
```

Unmatched tokens

If a token is unmatched during check preparation, the agent check handler will return an error, and the check will not be executed. Unmatched token errors are similar to this example:

```
error: unmatched token: template: :1:22: executing "" at <.system.hostname>: map has no entry for key "System"
```

Check config token errors are logged by the agent and sent to Sensu backend message transport as check failures.

Token data type limitations

As part of the substitution process, Sensu converts all tokens to strings. This means that token substitution cannot be applied to any non-string values like numbers or Booleans, although it can be applied to strings that are nested inside objects and arrays.

For example, token substitution **cannot** be used for specifying a check interval because the interval attribute requires an *integer* value. Token substitution **can** be used for alerting thresholds because those values are included within the command *string*.

Business service monitoring SDK

***COMMERCIAL FEATURE**: Access business service monitoring (BSM) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

Sensu's business service monitoring (BSM) feature uses a dedicated SDK of JavaScript-based expressions that provide additional functionality. Use the BSM SDK to create custom JavaScript expressions with complex logic.

BSM SDK expressions are defined in rule templates, so they act in the context of determining whether aggregate data derived from a service component's selection of Sensu Go events should trigger a rule-based event. They always receive a single event and some information about that event, like `event.timestamp` or `event.check.interval`, and always return either `true` or `false`.

BSM SDK expressions are evaluated by the Otto JavaScript VM as JavaScript programs.

***NOTE**: Sensu query expressions also provide JavaScript functions for using nested parameters and custom functions to retrieve events from the event store.*

Syntax quick reference

operator	description
`===`	Identity
`!==`	Nonidentity
`==`	Equality
`!=`	Inequality

`&&`	Logical AND
`||`	Logical OR
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to

Specification

BSM SDK expressions are valid ECMAScript 5 (JavaScript) expressions that return either `true` or `false`.Other values are not allowed.If an expression returns a value besides `true` or `false`, the Sensu backend log will record an error and the filter will evaluate to `false`.

The BSM SDK allows you to to express rules for the number or percentage of events with critical, warning, OK, and unknown statuses.You can also configure expressions to ignore silenced events.

Custom functions

The Sensu BSM SDK includes two custom functions: `sensu.Count()` and `sensu.Percentage()`.

sensu.Count()

The custom function `sensu.Count()` returns the number of events with the specified status.For example, to return the number of events with `ok` status:

```
sensu.Count("ok")
```

sensu.Percentage()

The custom function `sensu.Percentage()` returns the percentage of events with the specified status.

For example, to return the percentage of events with `critical` status:

```
sensu.Percentage("critical")
```

Example BSM SDK expression

The following BSM SDK expression creates a critical event if at least 35% of events generated by the service component have `critical` status **or** creates a warning event if the service component generates one or more events with `warning` status:

```
if (sensu.Percentage("critical") >= 35) {
  event.check = {status: 2, output: "critical event"}
} else if (sensu.Count("warning") >= 1) {
  event.check = {status: 1, output: "warning event"}
}
```

Augment event data with check hooks

Check hooks are **commands** the Sensu agent runs in response to the result of check execution. The Sensu agent executes the appropriate configured hook command based on the exit status code of the check (for example, `1`).

Check hooks allow you to automate data collection that operators would routinely perform to investigate observability alerts, which frees up precious operator time. Although you can use check hooks for rudimentary auto-remediation tasks, they are intended to enrich observability event data.

Follow this guide to create a check hook that captures the process tree if a check returns a status of `2` (critical, not running). You'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition. To run the `nginx_service` check used as an example in this guide, you'll need a Sensu entity with the subscription `webserver`.

To add the `webserver` subscription to the entity the Sensu agent is observing, first find your agent entity name:

```
sensuctl entity list
```

The `ID` is the name of your entity.

Replace `<entity_name>` with the name of your agent entity in the following sensuctl command. Run:

```
sensuctl entity update <entity_name>
```

- For `Entity Class`, press enter.

- For `Subscriptions`, type `webserver` and press enter.

Confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Install and configure NGINX

The `nginx_service` check requires a running NGINX service, so you'll need to install and configure NGINX.

> ***NOTE**: You may need to install and update the EPEL repository with `sudo yum install epel-release` and `sudo yum update` before you can install NGINX.*

Install NGINX:

```
sudo yum install nginx
```

Enable and start the NGINX service:

```
systemctl enable nginx && systemctl start nginx
```

Verify that NGINX is serving webpages:

```
curl -sI http://localhost
```

The response should include `HTTP/1.1 200 OK` to indicate that NGINX processed your request as expected:

```
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Wed, 06 Oct 2021 19:35:14 GMT
Content-Type: text/html
Content-Length: 4833
Last-Modified: Fri, 16 May 2014 15:12:48 GMT
Connection: keep-alive
ETag: "xxxxxxxx-xxxx"
Accept-Ranges: bytes
```

With your NGINX service running, you can configure the webserver check.

Create a hook

Create a new hook that runs a specific command to capture the process tree:

```
sensuctl hook create process_tree  \
--command 'ps aux' \
--timeout 10
```

To confirm that the hook was added, run:

SHELL

```
sensuctl hook info process_tree --format yaml
```

SHELL

```
sensuctl hook info process_tree --format wrapped-json
```

The response will include the complete hook resource definition in the specified format:

YML

```
---
type: HookConfig
```

```
api_version: core/v2
metadata:
  name: process_tree
spec:
  command: ps aux
  runtime_assets: null
  stdin: false
  timeout: 10
```

JSON

```
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "process_tree"
  },
  "spec": {
    "command": "ps aux",
    "runtime_assets": null,
    "stdin": false,
    "timeout": 10
  }
}
```

Assign the hook to a check

> ***NOTE**: Before you proceed, make sure you have added the sensu/sensu-processes-check dynamic runtime asset and the `nginx_service` check from the Monitor server resources guide. The hook you create in this step relies on the `nginx_service` check.*

Now that you've created the `process_tree` hook, you can assign it to the `nginx_service` check. Setting the `type` to `critical` ensures that whenever the check command returns a critical status, Sensu executes the `process_tree` hook and adds the output to the resulting event data.

To assign the hook to your `nginx_service` check, run:

```
sensuctl check set-hooks nginx_service  \
--type critical \
--hooks process_tree
```

Examine the check definition to confirm that it includes the hook.Run:

SHELL

```
sensuctl check info nginx_service --format yaml
```

SHELL

```
sensuctl check info nginx_service --format wrapped-json
```

You should find the `process_tree` hook listed in the `check_hooks` array, within the `critical` array:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: nginx_service
spec:
  check_hooks:
  - critical:
    - process_tree
  command: |
        sensu-processes-check --search '[{"search_string": "nginx"}]'
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 15
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines: []
  proxy_entity_name: ""
  publish: true
```

```
  round_robin: false
  runtime_assets:
  - sensu-processes-check
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - webserver
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "nginx_service"
  },
  "spec": {
    "check_hooks": [
      {
        "critical": [
          "process_tree"
        ]
      }
    ],
    "command": "sensu-processes-check --search '[{\"search_string\":
\"nginx\"}]'\n",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu-processes-check"
```

```
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "webserver"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```


***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Simulate a critical event

After you confirm that the hook is attached to your check, stop the NGINX service to observe the check hook in action on the next check execution.

To manually generate a critical event for your `nginx_service` check, run:

```
systemctl stop nginx
```

When you stop the service, the check will generate a critical event. After a few moments, run:

```
sensuctl event list
```

The response should list the `nginx_service` check, returning a CRITICAL status (`2`):

```
   Entity       Check                    Output                  Status  Silenced       Timestamp
UUID
_____________ _____________
```

```
_____ _____
_____ _____ _____
 sensu-centos  nginx_service  CRITICAL | 0 >= 1 (found >= required) evaluated false for "nginx"  2  false  2021-11-08 17:02:04 +0000 UTC  xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
                    Status - CRITICAL
```

Validate the check hook

Verify that the check hook is behaving properly against a specific event with `sensuctl`. To view the check hook command result within an event, replace <entity_name> in the following command with the name of your entity and run:

SHELL

```
sensuctl event info <entity_name> nginx_service --format yaml
```

SHELL

```
sensuctl event info <entity_name> nginx_service --format wrapped-json
```

The check hook command result is available in the `hooks` array, within the `check` scope:

YML

```
check:
  ...
  hooks:
  - command: ps aux
    duration: 0.00747112
    executed: 1645555463
    issued: 0
    metadata:
      name: process_tree
    output: |
      USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
      sensu     17638  0.0  0.1 155452  1860 ?        R    18:44   0:00 ps aux
    ...
    runtime_assets: null
```

```
    status: 0
    stdin: false
    timeout: 10
    ...
```

JSON

```
{
  "check": {
    "...": "...",
    "hooks": [
      {
        "command": "ps aux",
        "duration": 0.00747112,
        "executed": 1645555463,
        "issued": 0,
        "metadata": {
          "name": "process_tree"
        },
        "output": "USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
COMMAND\nsensu    17638  0.0  0.1 155452  1860 ?        R    18:44   0:00 ps aux\n",
        "...": "...",
        "runtime_assets": null,
        "status": 0,
        "stdin": false,
        "timeout": 10
      }
    ],
    "...": "..."
  }
}
```

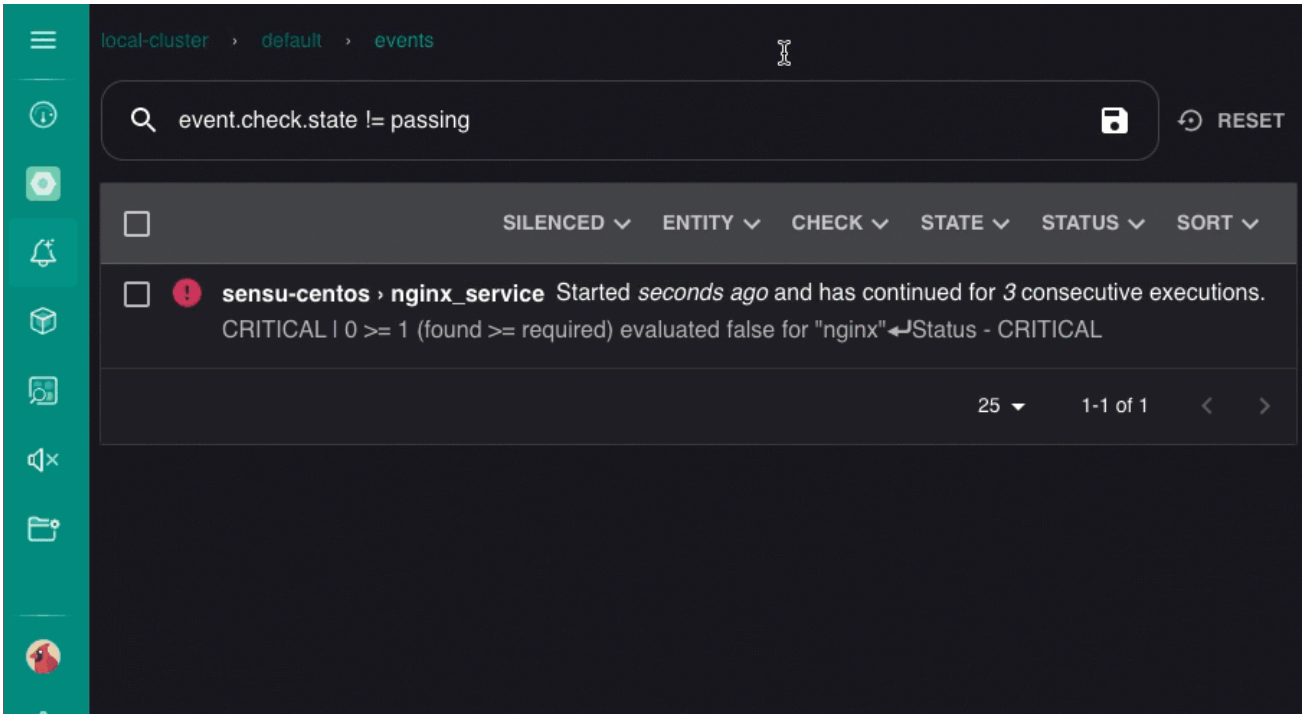
You can use sensuctl to query event info and send the response to `jq` so you can isolate the check hook output. In the following command, replace `<entity_name>` with the name of your entity and run:

```
sensuctl event info <entity_name> nginx_service --format json | jq -r
'.check.hooks[0].output'
```

This example output is truncated for brevity, but it reflects the output of the `ps aux` command specified in the check hook you created:

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.3  46164  6704 ?        Ss   Nov17   0:11
/usr/lib/systemd/systemd --switched-root --system --deserialize 20
root           2  0.0  0.0      0     0 ?        S    Nov17   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    Nov17   0:01 [ksoftirqd/0]
root           7  0.0  0.0      0     0 ?        S    Nov17   0:01 [migration/0]
root           8  0.0  0.0      0     0 ?        S    Nov17   0:00 [rcu_bh]
root           9  0.0  0.0      0     0 ?        S    Nov17   0:34 [rcu_sched]
```

You can also view check hook command results in the web UI.On the Events page, click the `nginx_service` event for your entity.Scroll down to the `HOOK` section and click it to expand and review hook command results.



Restart the NGINX service to clear the event:

```
systemctl start nginx
```

After a moment, you can verify that the event cleared:

```
sensuctl event list
```

The response should list the `nginx_service` check with an OK status (`0`).

Now when you are alerted that NGINX is not running, you can review the check hook output to confirm this is true with no need to start up an SSH session to investigate.

Next steps

To learn more about data collection with check hooks, read the hooks reference.

You can also create pipelines with event filters, mutators, and handlers to send the event data your checks generate to another service for analysis, tracking, and long-term storage. For example:

- Send data to Sumo Logic with Sensu
- Send PagerDuty alerts with Sensu
- Send Slack alerts with a pipeline

Collect Prometheus metrics with Sensu

The Prometheus ecosystem contains a number of actively maintained exporters, such as the node exporter for reporting hardware and operating system metrics or Google's cAdvisor exporter for monitoring containers. These exporters expose metrics that Sensu can collect and route to one or more time-series databases. Sensu and Prometheus can run in parallel, complementing each other and making use of environments where Prometheus is already deployed.

You can use the sensu/sensu-prometheus-collector dynamic runtime asset to create checks that collect metrics from a Prometheus exporter or the Prometheus query API. This allows Sensu to route the collected metrics to one or more time-series databases, such as InfluxDB or Graphite.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

The examples in this guide use CentOS 7 as the operating system, with all components running on the same compute resource. Commands and steps may change for different distributions or if components are running on different compute resources.

At the end of this guide, Prometheus will be scraping metrics. The check that uses the sensu/sensu-prometheus-collector asset will query the Prometheus API as a Sensu check and send the metrics to an InfluxDB Sensu handler, which will send metrics to an InfluxDB instance. Finally, Grafana will query InfluxDB to display the collected metrics.

Configure a Sensu entity

Use sensuctl to add an `app_tier` subscription to one of your entities. Before you run the following code, replace `<ENTITY_NAME>` with the name of the entity on your system.

***NOTE**: To find your entity name, run `sensuctl entity list`. The `ID` is the name of your entity.*

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.

- For `Subscriptions`, type `app_tier` and press enter.

Run this command to confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Install and configure Prometheus

Download and extract Prometheus with these commands:

```
wget https://github.com/prometheus/prometheus/releases/download/v2.6.0/prometheus-2.6.0.linux-amd64.tar.gz
```

```
tar xvfz prometheus-*.tar.gz
```

```
cd prometheus-*
```

Replace the default `prometheus.yml` configuration file with the following configuration:

```
global:
  scrape_interval: 15s
  external_labels:
    monitor: 'codelab-monitor'

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
```

Start Prometheus in the background:

```
nohup ./prometheus --config.file=prometheus.yml > prometheus.log 2>&1 &
```

Ensure Prometheus is running:

```
ps -ef | grep "[p]rometheus"
```

The response should be similar to this example:

```
vagrant   7647  3937  2 22:23 pts/0    00:00:00 ./prometheus --
config.file=prometheus.yml
```

Install and configure InfluxDB

Add an InfluxDB repo:

```
echo "[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key" | sudo tee
/etc/yum.repos.d/influxdb.repo
```

Install InfluxDB:

```
sudo yum -y install influxdb
```

Open `/etc/influxdb/influxdb.conf` and uncomment the `http` API line:

```
[http]
  # Determines whether HTTP endpoint is enabled.
  enabled = true
```

Start InfluxDB:

```
sudo systemctl start influxdb
```

Add the Sensu user and database with these commands:

```
influx -execute "CREATE DATABASE sensu"
```

```
influx -execute "CREATE USER sensu WITH PASSWORD 'sensu'"
```

```
influx -execute "GRANT ALL ON sensu TO sensu"
```

Install and configure Grafana

Install Grafana:

```
sudo yum install -y https://s3-us-west-2.amazonaws.com/grafana-
releases/release/grafana-5.1.4-1.x86_64.rpm
```

Change Grafana's listen port so that it does not conflict with the Sensu web UI:

```
sudo sed -i 's/^;http_port = 3000/http_port = 4000/' /etc/grafana/grafana.ini
```

Create a `/etc/grafana/provisioning/datasources/influxdb.yaml` file, and add an InfluxDB data

source:

```
apiVersion: 1

deleteDatasources:
  - name: InfluxDB
    orgId: 1

datasources:
  - name: InfluxDB
    type: influxdb
    access: proxy
    orgId: 1
    database: sensu
    user: grafana
    password: grafana
    url: http://localhost:8086
```

Start Grafana:

```
sudo systemctl start grafana-server
```

Create a Sensu InfluxDB handler

Add the Sensu InfluxDB handler asset

To add the sensu/sensu-influxdb-handler dynamic runtime asset to Sensu, run the following command:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.7.0 -r sensu-influxdb-handler
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-influxdb-handler:3.7.0
```

```
added asset: sensu/sensu-influxdb-handler:3.7.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-influxdb-handler"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `sensu-influxdb-handler`.

To confirm that the `sensu-influxdb-handler` asset is ready to use, run:

```
sensuctl asset list
```

The response should list the `sensu-influxdb-handler` dynamic runtime asset:

```
      Name                      URL                                                                  Hash
_____________________________ ____________________________________________________________________ _________

 sensu-influxdb-handler  //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_386.tar.gz          6719527
 sensu-influxdb-handler  //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_amd64.tar.gz        d05650d
 sensu-influxdb-handler  //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_armv7.tar.gz        38918c1
 sensu-influxdb-handler  //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_arm64.tar.gz        944075f
 sensu-influxdb-handler  //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_windows_amd64.tar.gz      8228cbc
 sensu-influxdb-handler  //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_darwin_amd64.tar.gz       7c73e1d
```

Add an InfluxDB handler

To add the handler definition that uses the Sensu InfluxDB Handler dynamic runtime asset, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Handler
```

```
api_version: core/v2
metadata:
  name: influxdb
spec:
  command: sensu-influxdb-handler -a 'http://127.0.0.1:8086' -d sensu -u sensu -p
sensu
  timeout: 10
  type: pipe
  runtime_assets:
  - sensu-influxdb-handler
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "influxdb"
  },
  "spec": {
    "command": "sensu-influxdb-handler -a 'http://127.0.0.1:8086' -d sensu -u sensu -
p sensu",
    "timeout": 10,
    "type": "pipe",
    "runtime_assets": [
      "sensu-influxdb-handler"
    ]
  }
}
EOF
```

PRO TIP: `sensuctl create --file` *also accepts files that contain multiple resources' definitions. You could save both the asset and handler definitions in a single file and use* `sensuctl create --file FILE_NAME.EXT` *to add them.*

Create a pipeline that includes the InfluxDB handler

Add your handler to a pipeline workflow. A single pipeline workflow can include one or more filters, one mutator, and one handler.

In this case, the pipeline includes only the InfluxDB handler you've already configured. To create the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: prometheus_metrics_workflows
spec:
  workflows:
  - name: influxdb_metrics
    handler:
      name: influxdb
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "prometheus_metrics_workflows"
  },
  "spec": {
    "workflows": [
      {
        "name": "influxdb_metrics",
        "handler": {
          "name": "influxdb",
          "type": "Handler",
```

```
            "api_version": "core/v2"
          }
        }
      ]
    }
  }
  EOF
```

Now you can add the `prometheus_metrics_workflows` pipeline to a check for check output metric extraction.

Collect Prometheus metrics with Sensu

Add the sensu/sensu-prometheus-collector asset

To add the sensu/sensu-prometheus-collector dynamic runtime asset to Sensu, run the following command:

```
sensuctl asset add sensu/sensu-prometheus-collector:1.3.2 -r sensu-prometheus-collector
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-prometheus-collector:1.3.2
added asset: sensu/sensu-prometheus-collector:1.3.2

You have successfully added the Sensu asset resource, but the asset will not get downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the appropriate
resource, populate the "runtime_assets" field with ["sensu-prometheus-collector"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `sensu-prometheus-collector`.

To confirm that the `sensu-prometheus-collector` asset is ready to use, run:

```
sensuctl asset list
```

The response should list the `sensu-prometheus-collector` dynamic runtime asset along with the previously added `sensu-influxdb-handler` asset:

```
      Name                        URL                                                                                  Hash
_____________________________ ____________________________________________________________________________________ _________

 sensu-influxdb-handler       //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_386.tar.gz             6719527
 sensu-influxdb-handler       //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_amd64.tar.gz           d05650d
 sensu-influxdb-handler       //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_armv7.tar.gz           38918c1
 sensu-influxdb-handler       //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_linux_arm64.tar.gz           944075f
 sensu-influxdb-handler       //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_windows_amd64.tar.gz         8228cbc
 sensu-influxdb-handler       //assets.bonsai.sensu.io/.../sensu-influxdb-handler_3.7.0_darwin_amd64.tar.gz          7c73e1d
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_windows_amd64.tar.gz   77f47c9
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_darwin_amd64.tar.gz    5e25a41
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_linux_armv7.tar.gz     2ae6727
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_linux_armv6.tar.gz     acad256
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_linux_arm64.tar.gz     6bfdbfc
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_linux_386.tar.gz       69e6d02
 sensu-prometheus-collector   //assets.bonsai.sensu.io/.../sensu-prometheus-collector_1.3.2_linux_amd64.tar.gz     aca56fa
```

Add a Sensu check that references the pipeline

To add the check definition that uses the sensu/sensu-prometheus-collector dynamic runtime asset and your `prometheus_metrics_workflows` pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
```

```
metadata:
  name: prometheus_metrics
spec:
  command: sensu-prometheus-collector -prom-url http://localhost:9090 -prom-query up
  handlers: []
  interval: 10
  publish: true
  output_metric_format: influxdb_line
  pipelines:
  - name: prometheus_metrics_workflows
    type: Pipeline
    api_version: core/v2
  subscriptions:
  - app_tier
  timeout: 0
  runtime_assets:
  - sensu-prometheus-collector
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "prometheus_metrics"
  },
  "spec": {
    "command": "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
query up",
    "handlers": [],
    "interval": 10,
    "publish": true,
    "output_metric_format": "influxdb_line",
    "pipelines": [
      {
        "name": "prometheus_metrics_workflows",
        "type": "Pipeline",
        "api_version": "core/v2"
      }
    ],
```

```
    "subscriptions": [
      "app_tier"
    ],
    "timeout": 0,
    "runtime_assets": [
      "sensu-prometheus-collector"
    ]
  }
}
EOF
```

The check subscription matches the subscription you added to your entity during set-up. The Sensu backend will coordinate check execution for you by comparing the subscriptions in your checks and entities. Sensu automatically executes a check when the check definition includes a subscription that matches a subscription for a Sensu entity.

Open the Sensu web UI to view the events generated by the `prometheus_metrics` check. Visit `http://127.0.0.1:3000`, and log in as the admin user (created during the initialization step when you installed the Sensu backend).

You can also view the metric event data using sensuctl. Run:

```
sensuctl event list
```

The response should be similar to this example:

```
    Entity          Check                          Output                                                                  Status   Silenced          Timestamp                    UUID
_____________ ____________________ ____________________________________________________________________________ ________ __________ ____________________________ ____________________________________
 sensu-centos   keepalive          Keepalive last sent from sensu-centos at 2022-01-14 15:23:00 +0000 UTC       0        false      2022-01-14 15:23:00 +0000 UTC   a9kr7kf8-21h8-459k-v6f8-ad93mf82mkfd
 sensu-centos   prometheus_metrics   up,instance=localhost:9090,job=prometheus value=1 1642173795                0        false      2022-01-14 15:23:15 +0000 UTC   sd8j4ls9-34gf-fr77-456g-92384738jd72
```

Visualize metrics with Grafana

Configure a dashboard in Grafana

Download the Grafana dashboard configuration file from the Sensu docs:

```
curl -O https://docs.sensu.io/sensu-go/latest/files/up_or_down_dashboard.json
```

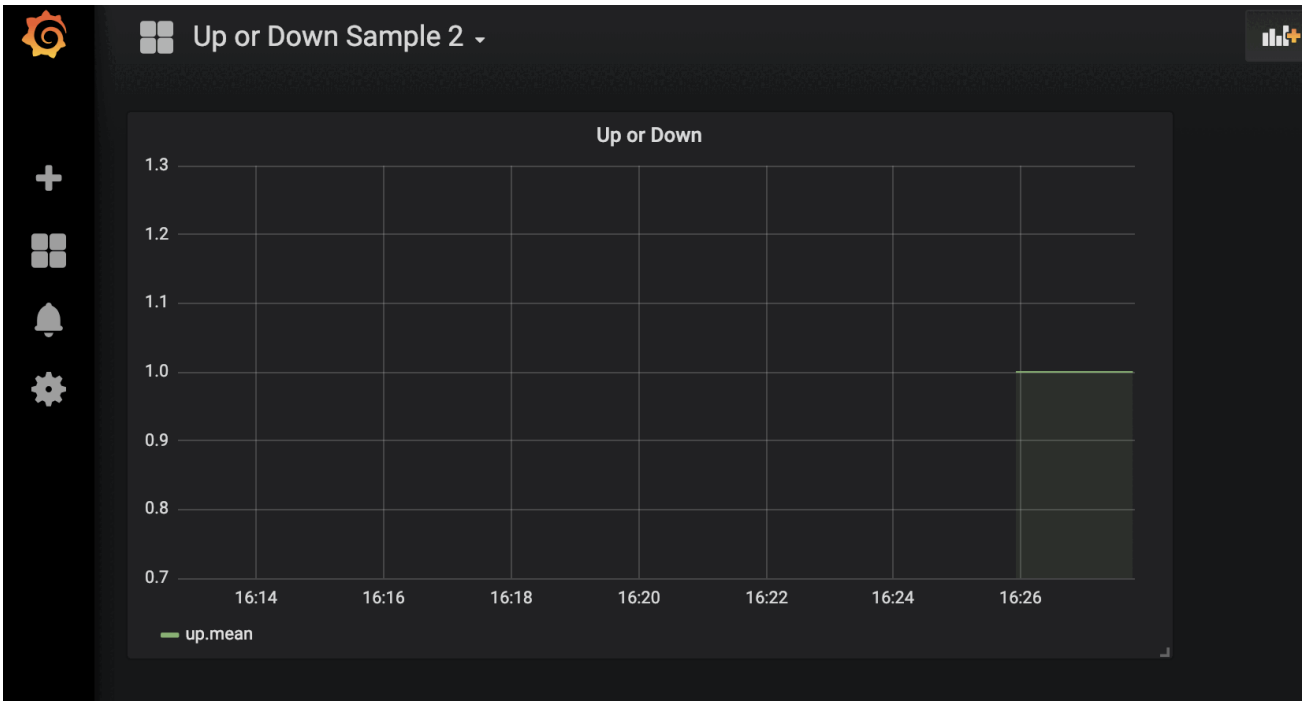
Using the downloaded file, add the dashboard to Grafana with an API call:

```
curl  -XPOST -H 'Content-Type: application/json' -d@up_or_down_dashboard.json
HTTP://admin:admin@127.0.0.1:4000/api/dashboards/db
```

View metrics in Grafana

Confirm metrics in Grafana: log in at `http://127.0.0.1:4000`. Use `admin` for both username and password.

Click **Home** in the upper left corner, then click the **Up or Down Sample 2** dashboard. The page should include a graph with initial metrics, similar to this example:



Next steps

You should now have a working observability pipeline with Prometheus scraping metrics. The sensu/sensu-prometheus-collector dynamic runtime asset runs via the `prometheus_metrics` Sensu check and collects metrics from the Prometheus API.

The check sends metrics to the `prometheus_metrics_workflows` pipeline, and the `influxdb` handler sends the metrics to InfluxDB. You can visualize the metrics in a Grafana dashboard.

Add the sensu/sensu-prometheus-collector to your Sensu ecosystem and include it in your monitoring as code repository. Use Prometheus to gather metrics and use Sensu to send them to the proper final destination. Prometheus has a comprehensive list of additional exporters to pull in metrics.

Collect service metrics with Sensu checks

> ***PRO TIP**: You can use the HTTP Service Monitoring (Local) integration in the Sensu Catalog to collect service metrics instead of following this guide. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Sensu checks are **commands** (or scripts) that the Sensu agent executes that output data and produce an exit code to indicate a state. If you are unfamiliar with checks, read the checks reference for details and examples. You can also learn how to configure monitoring checks in Monitor server resources.

This guide demonstrates how to use a check to extract service metrics for an NGINX webserver, with output in Nagios Performance Data format.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition. To run the NGINX webserver check, you'll need a Sensu entity with the subscription `webserver`.

To add the `webserver` subscription to the entity the Sensu agent is observing, first find your agent entity name:

```
sensuctl entity list
```

The `ID` is the name of your entity.

Replace `<ENTITY_NAME>` with the name of your agent entity in the following [sensuctl][17] command. Run:

```
```

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `webserver` and press enter.

Confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register the dynamic runtime asset

To power the check to collect service metrics, you will use a check in the sensu/http-checks dynamic runtime asset.Use sensuctl to register the sensu/http-checks dynamic runtime asset:

```
sensuctl asset add sensu/http-checks:0.5.0 -r http-checks
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/http-checks:0.5.0
added asset: sensu/http-checks:0.5.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["http-checks"].
```

This example uses the -r (rename) flag to specify a shorter name for the dynamic runtime asset: http-checks.

You can also download the dynamic runtime asset definition from Bonsai and register the asset with `sensuctl create --file filename.yml`.

Use sensuctl to confirm that both the http-checks dynamic runtime asset is ready to use:

```
sensuctl asset list
```

The sensuctl response should list http-checks:

```
   Name                URL                                                                  Hash
_____________ ____________________________________________________________________________ _________

 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_windows_amd64.tar.gz   52ae075
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_darwin_amd64.tar.gz    72d0f15
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_armv7.tar.gz     ef18587
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_arm64.tar.gz     3504ddf
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_386.tar.gz       60b8883
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_amd64.tar.gz     1db73a8
```

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Install and configure NGINX

The webserver check requires a running NGINX service, so you'll need to install and configure NGINX.

***NOTE**: You may need to install and update the EPEL repository with `sudo yum install epel-release` and `sudo yum update` before you can install NGINX.*

Install NGINX:

```
sudo yum install nginx
```

Enable and start the NGINX service:

```
systemctl enable nginx && systemctl start nginx
```

Verify that NGINX is serving webpages:

```
curl -sI http://localhost
```

The response should include `HTTP/1.1 200 OK` to indicate that NGINX processed your request as expected:

```
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Tue, 02 Nov 2021 20:15:40 GMT
Content-Type: text/html
Content-Length: 4833
Last-Modified: Fri, 16 May 2014 15:12:48 GMT
Connection: keep-alive
ETag: "xxxxxxxx-xxxx"
Accept-Ranges: bytes
```

With your NGINX service running, you can configure the check to collect service metrics.

***NOTE**: Read Monitor server resources with checks to learn how to monitor an NGINX webserver rather than collect metrics.*

Create a check to collect metrics

The http-checks dynamic runtime asset includes the `http-perf` check. To use this check, create the `collect-metrics` check with a command that uses `http-perf`:

```
sensuctl check create collect-metrics \
```

```
--command 'http-perf --url http://localhost --warning 1s --critical 2s' \
--interval 15 \
--subscriptions webserver \
--runtime-assets http-checks \
--output-metric-format nagios_perfdata
```

This example check specifies a 15-second interval for collecting metrics, a subscription to ensure the check will run on any entity that includes the `webserver` subscription, the name of the dynamic runtime asset the check needs to work properly, and the `nagios_perfdata` output metric format.

You should receive a confirmation response: `Created`.

To view the check resource you just created with sensuctl, run:

SHELL

```
sensuctl check info collect-metrics --format yaml
```

SHELL

```
sensuctl check info collect-metrics --format wrapped-json
```

The sensuctl response will list the complete check resource definition — you can add it to your monitoring as code repository:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: collect-metrics
spec:
  check_hooks: null
  command: http-perf --url http://localhost --warning 1s --critical 2s
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 15
  low_flap_threshold: 0
```

```
  output_metric_format: nagios_perfdata
  output_metric_handlers: null
  pipelines: []
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - http-checks
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - webserver
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "collect-metrics"
  },
  "spec": {
    "check_hooks": null,
    "command": "http-perf --url http://localhost --warning 1s --critical 2s",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "output_metric_format": "nagios_perfdata",
    "output_metric_handlers": null,
    "pipelines": [],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "http-checks"
    ],
```

```
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "webserver"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Confirm that your check is collecting metrics

If the check is collecting metrics correctly according to its `output_metric_format`, the metrics will be extracted in Sensu metric format and passed to the observability pipeline for handling. The Sensu agent will log errors if it cannot parse the check output.

Add a debug handler to write metric events to a file for inspection. To confirm that the check extracted metrics, inspect the event passed to the handler in the debug-event.json file. The event will include a top-level metrics section populated with metrics points arrays if the Sensu agent correctly ingested the metrics.

If you add the debug handler and configure the `collect-metrics` check to use it, the metrics event printed to the debug-event.json file will be similar to this example:

```
{
  "check": {
    "command": "http-perf --url http://localhost --warning 1s --critical 2s",
    "handlers": [
      "debug"
    ],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "publish": true,
```

```
    "runtime_assets": [
      "http-checks"
    ],
    "subscriptions": [
      "webserver"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 0.011235081,
    "executed": 1635886845,
    "history": [
      {
        "status": 0,
        "executed": 1635886785
      },
      {
        "status": 0,
        "executed": 1635886800
      },
      {
        "status": 0,
        "executed": 1635886815
      },
      {
        "status": 0,
        "executed": 1635886830
      },
      {
        "status": 0,
        "executed": 1635886845
      }
    ],
    "issued": 1635886845,
    "output": "http-perf OK: 0.001088s | dns_duration=0.000216,
tls_handshake_duration=0.000000, connect_duration=0.000140,
first_byte_duration=0.001071, total_request_duration=0.001088\n",
    "state": "passing",
```

```
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1635886845,
    "occurrences": 5,
    "occurrences_watermark": 5,
    "output_metric_format": "nagios_perfdata",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "collect-metrics",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "scheduler": "memory",
    "processed_by": "sensu-centos",
    "pipelines": []
  },
  "metrics": {
    "handlers": null,
    "points": [
      {
        "name": "dns_duration",
        "value": 0.000216,
        "timestamp": 1635886845,
        "tags": null
      },
      {
        "name": "tls_handshake_duration",
        "value": 0,
        "timestamp": 1635886845,
        "tags": null
      },
      {
        "name": "connect_duration",
        "value": 0.00014,
        "timestamp": 1635886845,
        "tags": null
      },
      {
        "name": "first_byte_duration",
        "value": 0.001071,
```

```
        "timestamp": 1635886845,
        "tags": null
      },
      {
        "name": "total_request_duration",
        "value": 0.001088,
        "timestamp": 1635886845,
        "tags": null
      }
    ]
  },
  "metadata": {
    "namespace": "default"
  },
  "id": "d19ee7f9-8cc5-447b-9059-895e89e14667",
  "sequence": 146,
  "pipelines": null,
  "timestamp": 1635886845,
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.9.2009",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::20b8:8cea:fa4:2e57/64"
            ]
```

```
      },
      {
        "name": "eth1",
        "mac": "08:00:27:40:ab:31",
        "addresses": [
          "192.168.200.95/24",
          "fe80::a00:27ff:fe40:ab31/64"
        ]
      }
    ]
  },
  "arch": "amd64",
  "libc_type": "glibc",
  "vm_system": "vbox",
  "vm_role": "guest",
  "cloud_provider": "",
  "processes": null
},
"subscriptions": [
  "webserver",
  "entity:sensu-centos"
],
"last_seen": 1635886845,
"deregister": false,
"deregistration": {},
"user": "agent",
"redact": [
  "password",
  "passwd",
  "pass",
  "api_key",
  "api_token",
  "access_key",
  "secret_key",
  "private_key",
  "secret"
],
"metadata": {
  "name": "sensu-centos",
  "namespace": "default"
},
"sensu_agent_version": "6.5.4"
```

```
    }
  }
```

Next step: Send metrics to a handler

Now that you know how to extract metrics from check output, learn to use a metrics handler to populate service and time-series metrics in InfluxDB. For a turnkey experience with the Sensu InfluxDB Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in InfluxDB.

Read the pipelines reference for information about configuring observability event processing workflows with event filters, mutators, and handlers.

You can also learn to use Sensu to collect Prometheus metrics.

Monitor Business Services

***COMMERCIAL FEATURE**: Access business service monitoring (BSM) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

Sensu's business service monitoring (BSM) provides high-level visibility into the current health of any number of your business services. Use BSM to monitor every component in your system with a top-down approach that produces meaningful alerts, prevents alert fatigue, and helps you focus on your core business services.

BSM requires two resources that work together to achieve top-down monitoring: service components and rule templates. Service components are the elements that make up your business services. Rule templates define the monitoring rules that produce events for service components based on customized evaluation expressions.

An example of a business service might be a company website. The website itself might have three service components: the primary webserver that publishes website pages, a backup webserver in case the primary webserver fails, and an inventory database for the shop section of the website. At least one webserver and the database must be in an OK state for the website to be fully available.

In this scenario, you could use BSM to create a current status page for this company website that displays the website's high-level status at a glance. As long as one webserver and the database have an OK status, the website status is OK. Otherwise, the website status is not OK. Most people probably just want to know whether the website is currently available — it won't matter to them whether the website is functioning with one or both webservers.

At the same time, the company *does* want to make sure the right person addresses any webserver failures, even if the website is technically still OK. BSM allows you to customize rule templates that apply a threshold for taking action for different service components as well as what action to take.

To continue the company website example, if the primary webserver fails but the backup webserver does not, you might use a rule template that creates a service ticket to address the next workday (in addition to the rule template that is emitting "OK" events for the current status page). Another monitoring rule might trigger an alert to the on-call operator should both webservers or the inventory database fail.

***NOTE**: BSM requires high event throughput. Configure a PostgreSQL datastore to achieve the required throughput and use the BSM feature.*

Service component example

Here is an example service component definition that includes the `website-services` service and applies the built-in `aggregate` rule template for events generated by checks with the `webserver` subscription:

YML

```
---
type: ServiceComponent
api_version: bsm/v1
metadata:
  name: webservers
spec:
  services:
    - website-services
  interval: 60
  query:
    - type: fieldSelector
      value: webserver in event.check.subscriptions
  rules:
    - template: aggregate
      name: webservers_50-70
      arguments:
        critical_threshold: 70
        warning_threshold: 50
  handlers:
    - slack
```

JSON

```
{
  "type": "ServiceComponent",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "webservers"
```

```
  },
  "spec": {
    "services": [
      "website-services"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "template": "aggregate",
        "name": "webservers_50-70",
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        }
      }
    ],
    "handlers": [
      "slack"
    ]
  }
}
```

Rule template example

This example lists the definition for the built-in aggregate rule template:

YML

```
---
type: RuleTemplate
api_version: bsm/v1
metadata:
  name: aggregate
  namespace: default
```

```
spec:
  arguments:
    properties:
      critical_count:
        description: create an event with a critical status if there the number of
          critical events is equal to or greater than this count
        type: number
      critical_threshold:
        description: create an event with a critical status if the percentage of
non-zero
          events is equal to or greater than this threshold
        type: number
      metric_handlers:
        default: {}
        description: metric handlers to use for produced metrics
        items:
          type: string
        type: array
      produce_metrics:
        default: {}
        description: produce metrics from aggregate data and include them in the
produced
          event
        type: boolean
      set_metric_annotations:
        default: {}
        description: annotate the produced event with metric annotations
        type: boolean
      warning_count:
        description: create an event with a warning status if there the number of
          critical events is equal to or greater than this count
        type: number
      warning_threshold:
        description: create an event with a warning status if the percentage of non-
zero
          events is equal to or greater than this threshold
        type: number
    required:
  description: Monitor a distributed service - aggregate one or more events into a
    single event. This BSM rule template allows you to treat the results of multiple
    disparate check executions - executed across multiple disparate systems - as a
    single event. This template is extremely useful in dynamic environments and/or
```

```
    environments that have a reasonable tolerance for failure. Use this template
when
    a service can be considered healthy as long as a minimum threshold is satisfied
    (e.g. at least 5 healthy web servers? at least 70% of N processes healthy?).
  eval: |2
    if (events && events.length == 0) {
        event.check.output = "WARNING: No events selected for aggregate
    ";
        event.check.status = 1;
        return event;
    }
    event.annotations["io.sensu.bsm.selected_event_count"] = events.length;
    percentOK = sensu.PercentageBySeverity("ok");
    if (!!args["produce_metrics"]) {
        var ts = Math.floor(new Date().getTime() / 1000);
        event.timestamp = ts;
        var tags = [
            {
                name: "service",
                value: event.entity.name
            },
            {
                name: "entity",
                value: event.entity.name
            },
            {
                name: "check",
                value: event.check.name
            }
        ];
        event.metrics = sensu.NewMetrics({
            points: [
                {
                    name: "percent_non_zero",
                    timestamp: ts,
                    value: sensu.PercentageBySeverity("non-zero"),
                    tags: tags
                },
                {
                    name: "percent_ok",
                    timestamp: ts,
                    value: percentOK,
```

```
            tags: tags
        },
        {
            name: "percent_warning",
            timestamp: ts,
            value: sensu.PercentageBySeverity("warning"),
            tags: tags
        },
        {
            name: "percent_critical",
            timestamp: ts,
            value: sensu.PercentageBySeverity("critical"),
            tags: tags
        },
        {
            name: "percent_unknown",
            timestamp: ts,
            value: sensu.PercentageBySeverity("unknown"),
            tags: tags
        },
        {
            name: "count_non_zero",
            timestamp: ts,
            value: sensu.CountBySeverity("non-zero"),
            tags: tags
        },
        {
            name: "count_ok",
            timestamp: ts,
            value: sensu.CountBySeverity("ok"),
            tags: tags
        },
        {
            name: "count_warning",
            timestamp: ts,
            value: sensu.CountBySeverity("warning"),
            tags: tags
        },
        {
            name: "count_critical",
            timestamp: ts,
            value: sensu.CountBySeverity("critical"),
```

```
                    tags: tags
                },
                {
                    name: "count_unknown",
                    timestamp: ts,
                    value: sensu.CountBySeverity("unknown"),
                    tags: tags
                }
            ]
        });
        if (!!args["metric_handlers"]) {
            event.metrics.handlers = args["metric_handlers"].slice();
        }
        if (!!args["set_metric_annotations"]) {
            var i = 0;
            while(i \u003c event.metrics.points.length) {
                event.annotations["io.sensu.bsm.selected_event_" +
event.metrics.points[i].name] = event.metrics.points[i].value.toString();
                i++;
            }
        }
    }
    if (!!args["critical_threshold"] && percentOK \u003c=
args["critical_threshold"]) {
        event.check.output = "CRITICAL: Less than " +
args["critical_threshold"].toString() + "% of selected events are OK (" +
percentOK.toString() + "%)
    ";
        event.check.status = 2;
        return event;
    }
    if (!!args["warning_threshold"] && percentOK \u003c= args["warning_threshold"])
{
        event.check.output = "WARNING: Less than " +
args["warning_threshold"].toString() + "% of selected events are OK (" +
percentOK.toString() + "%)
    ";
        event.check.status = 1;
        return event;
    }
    if (!!args["critical_count"]) {
        crit = sensu.CountBySeverity("critical");
```

```
        if (crit \u003e= args["critical_count"]) {
            event.check.output = "CRITICAL: " + args["critical_count"].toString() +
" or more selected events are in a critical state (" + crit.toString() + ")
    ";
            event.check.status = 2;
            return event;
        }
    }
    if (!!args["warning_count"]) {
        warn = sensu.CountBySeverity("warning");
        if (warn \u003e= args["warning_count"]) {
            event.check.output = "WARNING: " + args["warning_count"].toString() + "
or more selected events are in a warning state (" + warn.toString() + ")
    ";
            event.check.status = 1;
            return event;
        }
    }
    event.check.output = "Everything looks good (" + percentOK.toString() + "% OK)";
    event.check.status = 0;
    return event;
```

JSON

```
{
  "type": "RuleTemplate",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "aggregate",
    "namespace": "default"
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a critical status if the percentage
of non-zero events is equal to or greater than this threshold",
```

```
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers to use for produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from aggregate data and include them in
the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a warning status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event with a warning status if the percentage of
non-zero events is equal to or greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service - aggregate one or more events
into a single event. This BSM rule template allows you to treat the results of
multiple disparate check executions - executed across multiple disparate systems -
as a single event. This template is extremely useful in dynamic environments and/or
environments that have a reasonable tolerance for failure. Use this template when a
service can be considered healthy as long as a minimum threshold is satisfied (e.g.
at least 5 healthy web servers? at least 70% of N processes healthy?).",
    "eval": "\nif (events \\u0026\\u0026 events.length == 0) {\n
```

```
event.check.output = \"WARNING: No events selected for aggregate\n\";\n
event.check.status = 1;\n    return
event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_event_count\"] =
events.length;\n\npercentOK = sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var ts = Math.floor(new Date().getTime() /
1000);\n\n    event.timestamp = ts;\n\n    var tags = [\n        {\n
name: \"service\",\n            value: event.entity.name\n        },\n        {\n
name: \"entity\",\n            value: event.entity.name\n        },\n        {\n
name: \"check\",\n            value: event.check.name\n        }\n    ];\n\n
event.metrics = sensu.NewMetrics({\n        points: [\n            {\n
name: \"percent_non_zero\",\n                timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"non-zero\"),\n                tags: tags\n
},\n            {\n                name: \"percent_ok\",\n                timestamp:
ts,\n                value: percentOK,\n                tags: tags\n            },\n
{\n                name: \"percent_warning\",\n                timestamp: ts,\n
value: sensu.PercentageBySeverity(\"warning\"),\n                tags: tags\n
},\n            {\n                name: \"percent_critical\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n                name:
\"percent_unknown\",\n                timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"unknown\"),\n                tags: tags\n
},\n            {\n                name: \"count_non_zero\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n                name: \"count_ok\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"ok\"),\n
tags: tags\n            },\n            {\n                name:
\"count_warning\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"warning\"),\n                tags: tags\n            },\n
{\n                name: \"count_critical\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"critical\"),\n                tags: tags\n
},\n            {\n                name: \"count_unknown\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n            }\n        ]\n    });\n\n    if
(!!args[\"metric_handlers\"]) {\n        event.metrics.handlers =
args[\"metric_handlers\"].slice();\n    }\n\n    if
(!!args[\"set_metric_annotations\"]) {\n        var i = 0;\n\n        while(i
\\u003c event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n            i++;\n        }\n
}\n}\n\nif (!!args[\"critical_threshold\"] \\u0026\\u0026 percentOK \\u003c=
args[\"critical_threshold\"]) {\n    event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected events are OK (\" +
```

```
percentOK.toString() + \"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] \\u0026\\u0026 percentOK \\u003c=
args[\"warning_threshold\"]) {\n    event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit \\u003e=
args[\"critical_count\"]) {\n        event.check.output = \"CRITICAL: \" +
args[\"critical_count\"].toString() + \" or more selected events are in a critical
state (\" + crit.toString() + \")\n\";\n        event.check.status = 2;\n
return event;\n    }\n}\n\nif (!!args[\"warning_count\"]) {\n    warn =
sensu.CountBySeverity(\"warning\");\n\n    if (warn \\u003e=
args[\"warning_count\"]) {\n        event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected events are in a warning
state (\" + warn.toString() + \")\n\";\n        event.check.status = 1;\n
return event;\n    }\n}\n\nevent.check.output = \"Everything looks good (\" +
percentOK.toString() + \"% OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}
```

Configure BSM via the web UI

The Sensu web UI BSM module allows you to create, edit, and delete service components and rule templates inside the web UI.

Configure BSM via APIs and sensuctl

BSM service components and rule templates are Sensu resources with complete definitions, so you can use Sensu's service component and rule template APIs to create, retrieve, update, and delete service components and rule templates.

You can also use sensuctl to create and manage service components and rule templates via the APIs from the command line.

Monitor server resources with checks

Sensu checks are commands (or scripts) the Sensu agent executes that output data and produce an exit code to indicate a state.

You can use checks to monitor server resources (for example, to learn how much disk space you have left), services, and application health (for example, to check whether NGINX is running) and collect and analyze metrics. This guide includes two check examples to help you monitor server resources (specifically, CPU usage and NGINX status).

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition. To run the CPU and NGINX webserver checks, you'll need a Sensu entity with the subscriptions `system` and `webserver`.

> ***NOTE**: In production, your CPU and NGINX servers would be different entities, with the `system` subscription specified for the CPU entity and the `webserver` subscription specified for the NGINX entity. To keep things streamlined, this guide uses one entity to represent both.*

To add the `system` and `webserver` subscriptions to the entity the Sensu agent is observing, first find your agent entity name:

```
sensuctl entity list
```

The `ID` is the name of your entity.

Replace `<ENTITY_NAME>` with the name of your agent entity in the following sensuctl command. Run:

```
```

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `system,webserver` and press enter.

Confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register dynamic runtime assets

You can write shell scripts in the `command` field of your check definitions, but we recommend using existing check plugins instead. Check plugins must be available on the host where the agent is running for the agent to execute the check. This guide uses dynamic runtime assets to manage plugin installation.

Register the sensu/check-cpu-usage asset

The sensu/check-cpu-usage dynamic runtime asset includes the `check-cpu-usage` command, which your CPU check will rely on.

To register the sensu/check-cpu-usage dynamic runtime asset, run:

```
sensuctl asset add sensu/check-cpu-usage:0.2.2 -r check-cpu-usage
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/check-cpu-usage:0.2.2
added asset: sensu/check-cpu-usage:0.2.2

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
```

```
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["check-cpu-usage"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `check-cpu-usage`.

You can also download dynamic runtime asset definitions from Bonsai and register the asset with `sensuctl create --file filename.yml`.

Register the sensu/sensu-processes-check asset

Then, use this command to register the sensu/sensu-processes-check dynamic runtime asset, which you'll use later for your webserver check:

```
sensuctl asset add sensu/sensu-processes-check:0.2.0 -r sensu-processes-check
```

To confirm that both dynamic runtime assets are ready to use, run:

```
sensuctl asset list
```

The response should list the renamed check-cpu-usage and sensu-processes-check dynamic runtime assets:

```
      Name                        URL                                                                  Hash
_____________________ _____________________________________________________________________ _________
 check-cpu-usage      //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_windows_amd64.tar.gz      900cfdf
 check-cpu-usage      //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_darwin_amd64.tar.gz       db81ee7
 check-cpu-usage      //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_armv7.tar.gz        400aacc
 check-cpu-usage      //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_arm64.tar.gz        bef7802
 check-cpu-usage      //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_386.tar.gz          a2dcb53
 check-cpu-usage      //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_amd64.tar.gz        2453973
 sensu-processes-check //assets.bonsai.sensu.io/.../sensu-processes-check_0.2.0_windows_amd64.tar.gz 42e2d71
```

```
sensu-processes-check  //assets.bonsai.sensu.io/.../sensu-processes-check_0.2.0_darwin_amd64.tar.gz  957c008
sensu-processes-check  //assets.bonsai.sensu.io/.../sensu-processes-check_0.2.0_linux_armv7.tar.gz  20cc5b1
sensu-processes-check  //assets.bonsai.sensu.io/.../sensu-processes-check_0.2.0_linux_arm64.tar.gz  c68b5f0
sensu-processes-check  //assets.bonsai.sensu.io/.../sensu-processes-check_0.2.0_linux_386.tar.gz  4c47caa
sensu-processes-check  //assets.bonsai.sensu.io/.../sensu-processes-check_0.2.0_linux_amd64.tar.gz  70e830f
```

Because plugins are published for multiple platforms, including Linux and Windows, the output will include multiple entries for each of the dynamic runtime assets.

> ***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Create a check to monitor a server

Now that the dynamic runtime assets are registered, create a check named `check_cpu` that runs the command `check-cpu-usage -w 75 -c 90` with the check-cpu-usage dynamic runtime asset at an interval of 60 seconds for all entities subscribed to the `system` subscription. This check generates a warning event (`-w`) when CPU usage reaches 75% and a critical alert (`-c`) at 90%.

```
sensuctl check create check_cpu \
--command 'check-cpu-usage -w 75 -c 90' \
--interval 60 \
--subscriptions system \
--runtime-assets check-cpu-usage
```

You should receive a confirmation message:

```
Created
```

To view the complete resource definition for `check_cpu` , run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The sensuctl response will include the complete `check_cpu` resource definition in the specified format:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines: []
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```


If you want to share, reuse, and maintain this check just like you would code, you can save it to a file and start building a monitoring as code repository.

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Validate the CPU check

The Sensu agent uses WebSocket to communicate with the Sensu backend, sending event data as JSON messages. As your checks run, the Sensu agent captures check standard output (stdout) or standard error (stderr). This data will be included in the JSON payload the agent sends to your Sensu backend as the event data.

It might take a few moments after you create the check for the check to be scheduled on the entity and the event to return to the Sensu backend. Use sensuctl to view the event data and confirm that Sensu is monitoring CPU usage:

```
sensuctl event list
```

The response should list the `check_cpu` check, returning an OK status (`0`)

```
    Entity       Check                                                      Output
Status   Silenced          Timestamp                        UUID
────────────────── ─────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────── ───────── ────────── ─────────────────────────────
────────────────────────────────────────
  sensu-centos   check_cpu   check-cpu-usage OK: 1.02% CPU usage | cpu_idle=98.98, cpu_system=0.51, cpu_user=0.51, cpu_nice=0.00, cpu_iowait=0.00, cpu_irq=0.00, cpu_softirq=0.00, cpu_steal=0.00, cpu_guest=0.00, cpu_guestnice=0.00
0   false      2021-10-06 19:25:43 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Create a check to monitor a webserver

In this section, you'll create a check to monitor an NGINX webserver, similar to the CPU check you created in the previous section but using the `webserver` subscription rather than `system`.

Install and configure NGINX

The webserver check requires a running NGINX service, so you'll need to install and configure NGINX.

***NOTE**: You may need to install and update the EPEL repository with `sudo yum install epel-release` and `sudo yum update` before you can install NGINX.*

Install NGINX:

```
sudo yum install nginx
```

Enable and start the NGINX service:

```
systemctl enable nginx && systemctl start nginx
```

Verify that NGINX is serving webpages:

```
curl -sI http://localhost
```

The response should include `HTTP/1.1 200 OK` to indicate that NGINX processed your request as expected:

```
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Wed, 06 Oct 2021 19:35:14 GMT
Content-Type: text/html
Content-Length: 4833
Last-Modified: Fri, 16 May 2014 15:12:48 GMT
Connection: keep-alive
ETag: "xxxxxxxx-xxxx"
Accept-Ranges: bytes
```

With your NGINX service running, you can configure the webserver check.

Create the webserver check definition

Create a check that uses `sensu-processes-check` in the command to search for the string `nginx`. The `nginx_service` check will run at an interval of 15 seconds and determine whether the `nginx` service is among the running processes for all entities subscribed to the `webserver` subscription.

To create the `nginx_service` check, run the following command:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: nginx_service
spec:
  command: >
    sensu-processes-check
    --search
    '[{"search_string": "nginx"}]'
  subscriptions:
  - webserver
  interval: 15
  publish: true
  runtime_assets:
  - sensu-processes-check
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "nginx_service"
  },
  "spec": {
    "command": "sensu-processes-check --search '[{\"search_string\":
\"nginx\"}]'\n",
    "subscriptions": [
      "webserver"
    ],
```

```
    "interval": 15,
    "publish": true,
    "runtime_assets": [
      "sensu-processes-check"
    ]
  }
}
EOF
```

You should receive a confirmation message:

```
Created
```

To view the complete resource definition for `nginx_service`, run:

SHELL

```
sensuctl check info nginx_service --format yaml
```

SHELL

```
sensuctl check info nginx_service --format wrapped-json
```

The sensuctl response will include the complete `nginx_service` resource definition in the specified format:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: nginx_service
spec:
  check_hooks: null
  command: |
        sensu-processes-check --search '[{"search_string": "nginx"}]'
```

```
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 15
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines: []
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu-processes-check
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - webserver
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "nginx_service"
  },
  "spec": {
    "check_hooks": null,
    "command": "sensu-processes-check --search '[{\"search_string\":
\"nginx\"}]'\n",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [],
    "proxy_entity_name": "",
```

```
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu-processes-check"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "webserver"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

As with the `check_cpu` check, you can share, reuse, and maintain this check just like code.

Validate the webserver check

It might take a few moments after you create the check for the check to be scheduled on the entity and the event to return to the Sensu backend.Use sensuctl to view event data and confirm that Sensu is monitoring the NGINX webserver status:

```
sensuctl event list
```

The response should list the `nginx_service` check, returning an OK status (`0`):

```
    Entity        Check                      Output                    Status   Silenced          Timestamp
UUID
_____________ _______________ ____________________________________ ________ __________ ____________________________
____________ ____________________________________ ____________________________________
 sensu-centos   nginx_service   OK       | 2 >= 1 (found >= required) evaluated true for "nginx"          0   false     2021-11-
08 16:59:34 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
                        Status - OK
```

Simulate a critical event

To manually generate a critical event for your `nginx_service` check, stop the NGINX service. Run:

```
systemctl stop nginx
```

When you stop the service, the check will generate a critical event. After a few moments, run:

```
sensuctl event list
```

The response should list the `nginx_service` check, returning a CRITICAL status (`2`):

```
    Entity         Check                          Output                                 Status   Silenced          Timestamp
UUID
_____________ _____________ ____________________________________________________________ ________ _________ ____________________ ____________________________________
 sensu-centos   nginx_service   CRITICAL | 0 >= 1 (found >= required) evaluated false for "nginx"          2   false     2021-
11-08 17:02:04 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
                        Status - CRITICAL
```

Restart the NGINX service to clear the event:

```
systemctl start nginx
```

After a moment, you can verify that the event cleared:

```
sensuctl event list
```

The response should list the `nginx_service` check with an OK status (`0`).

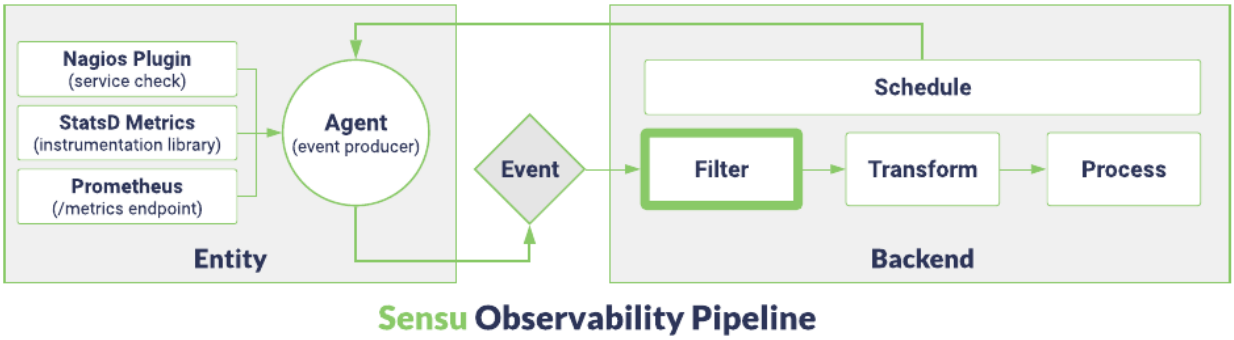
Next steps

Now that you know how to create checks to monitor CPU usage and NGINX webserver status, read the checks reference and assets reference for more detailed information.Or, learn how to monitor external resources with proxy checks and entities.

You can also create pipelines to send alerts to email, PagerDuty, or Slack based on the status events your checks are generating.

Read the pipelines reference for information about configuring observability event processing workflows with event filters, mutators, and handlers.

Filter your observation data



Sensu Observability Pipeline

Next **or click any element in the pipeline to jump to it.**

In the filter stage, Sensu executes event filters.

The filter stage of the Sensu observability pipeline applies the conditions, triggers, and thresholds you specify in your event filter defintions to the events your checks generate. Event filters give you control over which events continue through your pipeline and become alerts. For example, use the built-in is_incident event filter to allow only high-priority events through your Sensu pipeline and reduce noise for operators.

To tell Sensu which event filters you want to apply, you list them in your pipeline definitions. Sensu compares your observation data in events against the expressions in your event filters to determine whether each event should continue through the pipeline or be removed. Event filters can be inclusive or exclusive, so you can require events to match or not match your filter expressions.

Here's an example that shows the resource definition for an event filter that would allow handling for only events with the custom entity label `"region": "us-west-1"`:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: production_filter
spec:
  action: allow
```

```
  expressions:
  - event.entity.labels['region'] == 'us-west-1'
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "production_filter"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.entity.labels['region'] == 'us-west-1'"
    ]
  }
}
```

Sensu applies event filters in the order that they are listed in your pipeline definition. Any events that the filters do not remove from your pipeline will be processed according to your handler configuration.

As soon as an event filter removes an event from your pipeline because it does not meet the conditions, triggers, or thresholds you specified, the Sensu observability pipeline ceases analysis for the event. Sensu will not transform or process events that your event filter removes from your pipeline.

Use Bonsai, the Sensu asset hub, to discover, download, and share Sensu event filter dynamic runtime assets. Read Use assets to install plugins to get started.

Event filters reference

Sensu executes event filters during the **filter** stage of the observability pipeline.

Sensu event filters are applied when you configure a pipeline with a workflow that uses one or more filters. Before executing the handler in a pipeline workflow, the Sensu backend will apply any event filters listed in the same pipeline workflow to the observation data in events. If the filters do not remove the event, the handler will be executed.

The event filter analysis performs these steps:

- When the Sensu backend is processing an event, it checks for filters in the pipeline (or pipelines) specified in the event's check definition. Before executing any handlers listed in the pipeline, Sensu applies any event filters and mutators listed in the pipeline.
- If multiple filters are configured for a pipeline, they are executed sequentially.
- Filter `expressions` are compared with event data.

Event filters can be inclusive (only matching events are handled) or exclusive (matching events are not handled). Read Inclusive and exclusive event filters for details.

As soon as a filter removes an event, no further analysis is performed and the pipeline workflow's handler will not be executed.

Event filter example (minimum required attributes)

This example shows the minimum required attributes for an event filter resource:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: filter_minimum
spec:
  action: allow
```

```
  expressions:
  - event.check.occurrences == 1
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "filter_minimum"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ]
  }
}
```

Inclusive and exclusive event filters

Event filters can be *inclusive* or *exclusive*:

- In *inclusive* filtering, only events that match the defined filter expressions will be handled.
- In *exclusive* filtering, events will be handled only if they do not match the defined filter expressions.

Use the `action` attribute in the event filter definition to control whether the filter is inclusive or exclusive:

- To use inclusive filtering, set the `action` attribute to `allow` in the event filter definition (`"action": "allow"`).
- To use exclusive filtering, set the `action` attribute to `deny` in the event filter definition (`"action": "deny"`).

Multiple inclusive or exclusive event filters

Multiple *inclusive* event filters are equivalent to using an `AND` query operator: Sensu will only handle events if they match *all* of the inclusive filters (`x AND y AND z`).

Multiple *exclusive* event filters are equivalent to using an `OR` operator: Sensu will only handle events if they *don't* match *any* of the exclusive filters (`x OR y OR z`).

Filter expression comparison

Event filter expressions are compared directly with their event data counterparts.

- For *inclusive* event filter definitions (`"action": "allow"`), matching expressions will result in the filter returning a `true` value. The event will pass through the filter and continue to be processed with additional filters (if defined), mutators (if defined), and handlers.
- For *exclusive* event filter definitions (`"action": "deny"`), matching expressions will result in the filter returning a `false` value, and the event will not pass through the filter.

Filter expression evaluation

When more complex conditional logic is needed than direct filter expression comparison, Sensu event filters provide support for expression evaluation using Otto.Otto is an ECMAScript 5 (JavaScript) virtual machine that evaluates JavaScript expressions provided in an event filter.

In event filter expressions, place string values inside single or double quotes.These filter expressions are equivalent in EMCAScript:

```
event.check.annotations['service_priority'] == 1
event.check.annotations["service_priority"] == 1
```

There are some caveats to using Otto: not all of the regular expressions (regex) specified in ECMAScript 5 will work.Review the Otto README for more details.

Use Go regex syntax to create event filter expressions that combine any available event, check, or entity attributes with `match(<regex>)` .For example, this event filter allows handling for events whose `event.check.name` ends with `metrics` :

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: metrics-checks-only
spec:
  action: allow
  expressions:
  - event.check.name.match(/metrics$/)
```

JSON

```
{
   "type": "EventFilter",
   "api_version": "core/v2",
   "metadata": {
      "name": "metrics-checks-only"
   },
   "spec": {
      "action": "allow",
      "expressions": [
         "event.check.name.match(/metrics$/)"
      ]
   }
}
```

Here's another example that uses regex matching for event entity labels. This event filter allows handling for events created by entities with the `region` label `us-west-1`, `us-west-2`, or `us-west-3`:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: us-west-events
spec:
  action: allow
  expressions:
```

```
  - event.entity.labels.region.match(/us-west-\b[1-3]\b/)
```

JSON

```
{
   "type": "EventFilter",
   "api_version": "core/v2",
   "metadata": {
      "name": "us-west-events"
   },
   "spec": {
      "action": "allow",
      "expressions": [
         "event.entity.labels.region.match(/us-west-\b[1-3]\b/)"
      ]
   }
}
```

Filter dynamic runtime assets

Sensu event filters can include dynamic runtime assets in their execution context. When valid dynamic runtime assets are associated with an event filter, Sensu evaluates any files it finds that have a `.js` extension before executing the filter. The result of evaluating the scripts is cached for a given asset set for the sake of performance. For an example of how to implement an event filter as an asset, read Reduce alert fatigue.

Built-in event filters

Sensu includes built-in event filters to help you customize event pipelines for metrics and alerts. To start using built-in event filters, read Send Slack alerts and Plan maintenance.

> ***NOTE**: Sensu Go does not include the built-in occurrence-based event filter in Sensu Core 1.x, but you can replicate its functionality with the repeated events filter definition.*

Built-in filter: is_incident

The is_incident event filter is included in every installation of the Sensu backend. You can use the is_incident filter to allow only high-priority events through a Sensu pipeline. For example, you can use the is_incident filter to reduce noise when sending notifications to Slack. When applied to a pipeline workflow, the is_incident filter allows warning (`"status": 1`), critical (`"status": 2`), other (unknown or custom status), and resolution events to be processed.

To use the is_incident event filter, include `is_incident` in the pipeline `filters` object:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: incident_alerts
spec:
  workflows:
  - name: slack_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "slack_alerts",
        "filters": [
          {
```

```
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

The is_incident event filter applies the following filtering logic:

status	allow	discard
0		❌
1	✅	
2	✅	
other (unknown or custom status)	✅	
resolution event such as 1 –> 0 or 3 –> 0	✅	

Built-in filter: not_silenced

Sensu silencing lets you suppress handler execution on an on-demand basis so you can quiet incoming alerts and plan maintenance.

To allow silencing for a pipeline workflow, add `not_silenced` to the pipeline `filters` object:

YML

```yaml
---
type: Pipeline
api_version: core/v2
metadata:
  name: incident_alerts
spec:
  workflows:
  - name: slack_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2
```

JSON

```json
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "slack_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
```

```
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

When applied in a pipeline configuration, the not_silenced event filter silences events that include the `silenced` attribute. The pipeline in the example above uses both the not_silenced and is_incident event filters, preventing low-priority and silenced events from being sent to Slack.

Built-in filter: has_metrics

The has_metrics event filter is included in every installation of the Sensu backend. When applied in a pipeline workflow, the has_metrics filter allows only events that contain Sensu metrics to be processed. You can use the has_metrics filter to prevent handlers that require metrics from failing in case of an error in metric collection.

To use the has_metrics event filter, include `has_metrics` in the pipeline `filters` array:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: metrics_pipeline
spec:
  workflows:
  - name: influxdb_metrics
    filters:
    - name: has_metrics
      type: EventFilter
```

```
      api_version: core/v2
    handler:
      name: influxdb
      type: Handler
      api_version: core/v2
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "metrics_pipeline"
  },
  "spec": {
    "workflows": [
      {
        "name": "influxdb_metrics",
        "filters": [
          {
            "name": "has_metrics",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "influxdb",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

When applied in a pipeline configuration, the has_metrics event filter allows only events that include a `metrics` scope.

Build event filter expressions with Sensu query expressions

You can write custom event filter expressions as Sensu query expressions using the event data attributes described in this section. For more information about event attributes, read the event reference.

Syntax quick reference

operator	description
`===` / `!==`	Identity operator / Nonidentity operator
`==` / `!=`	Equality operator / Inequality operator
`&&` / `||`	Logical AND / Logical OR
`<` / `>`	Less than / Greater than
`<=` / `>=`	Less than or equal to / Greater than or equal to

Event attributes available to filters

attribute	type	description
`event.has_check`	Boolean	Returns true if the event contains check data
`event.has_metrics`	Boolean	Returns true if the event contains metrics
`event.is_incident`	Boolean	Returns true for critical alerts (status `2`), warnings (status `1`), and resolution events (status `0` transitioning from status `1` or `2`)
`event.is_resolution`	Boolean	Returns true if the event status is OK (`0`) and the previous event was of a non-zero status
`event.is_silenced`	Boolean	Returns true if the event matches an active silencing entry

`event.timestamp`	integer	Time that the event occurred in seconds since the Unix epoch

Check attributes available to filters

attribute	type	description
`event.check.annotations`	map	Custom annotations applied to the check
`event.check.command`	string	The command executed by the check
`event.check.cron`	string	Check execution schedule using cron syntax
`event.check.discard_output`	Boolean	Whether the check is configured to discard check output from event data
`event.check.duration`	float	Command execution time in seconds
`event.check.env_vars`	array	Environment variables used with command execution
`event.check.executed`	integer	Time that the check was executed in seconds since the Unix epoch
`event.check.handlers`	array	Sensu event handlers assigned to the check
`event.check.high_flap_threshold`	integer	The check's flap detection high threshold in percent state change
`event.check.history`	array	Check status history for the last 21 check executions
`event.check.hooks`	array	Check hook execution data
`event.check.interval`	integer	The check execution frequency in seconds

`event.check.issued`	integer	Time that the check request was issued in seconds since the Unix epoch
`event.check.labels`	map	Custom labels applied to the check
`event.check.last_ok`	integer	The last time that the check returned an OK status (`0`) in seconds since the Unix epoch
`event.check.low_flap_threshold`	integer	The check's flap detection low threshold in percent state change
`event.check.max_output_size`	integer	Maximum size of stored check outputs in bytes
`event.check.name`	string	Check name
`event.check.occurrences`	integer	The number of preceding events with the same status as the current event
`event.check.occurrences_watermark`	integer	For resolution events, the number of preceding events with a non-OK status
`event.check.output`	string	The output from the execution of the check command
`event.check.output_metric_format`	string	The metric format generated by the check command: `nagios_perfdata` , `graphite_plaintext` , `influxdb_line` , `opentsdb_line` , or `prometheus_text`
`event.check.output_metric_handlers`	array	Sensu metric handlers assigned to the check
`event.check.proxy_entity_name`	string	The entity name, used to create a proxy entity for an external resource
`event.check.proxy_requests`	map	Proxy request configuration
`event.check.publish`	Boolean	Whether the check is scheduled automatically
`event.check.round`	Bool	Whether the check is configured to be executed in a round-

`_robin`	ean	robin style
`event.check.runtime_assets`	array	Sensu dynamic runtime assets used by the check
`event.check.state`	string	The state of the check: `passing` (status `0`), `failing` (status other than `0`), or `flapping`
`event.check.status`	integer	Exit status code produced by the check: `0` (OK), `1` (warning), `2` (critical), or other status (unknown or custom status)
`event.check.stdin`	Boolean	Whether the Sensu agent writes JSON-serialized entity and check data to the command process' stdin
`event.check.subscriptions`	array	Subscriptions that the check belongs to
`event.check.timeout`	integer	The check execution duration timeout in seconds
`event.check.total_state_change`	integer	The total state change percentage for the check's history
`event.check.ttl`	integer	The time-to-live (TTL) until the event is considered stale, in seconds
`event.metrics.handlers`	array	Sensu metric handlers assigned to the check
`event.metrics.points`	array	Metrics data points including a name, timestamp, value, and tags

Entity attributes available to filters

attribute	type	description
`event.entity.annotations`	map	Custom annotations assigned to the entity
`event.entity.deregister`	Boolean	Whether the agent entity should be removed when it stops sending keepalive messages

`event.entity.deregistration`	map	A map that contains a handler name for use when an entity is deregistered
`event.entity.entity_class`	string	The entity type: usually `agent` or `proxy`
`event.entity.labels`	map	Custom labels assigned to the entity
`event.entity.last_seen`	integer	Timestamp the entity was last seen in seconds since the Unix epoch
`event.entity.name`	string	Entity name
`event.entity.redact`	array	List of items to redact from log messages
`event.entity.subscriptions`	array	List of subscriptions assigned to the entity
`event.entity.system`	map	Information about the entity's system
`event.entity.system.arch`	string	The entity's system architecture
`event.entity.system.hostname`	string	The entity's hostname
`event.entity.system.network`	map	The entity's network interface list
`event.entity.system.os`	string	The entity's operating system
`event.entity.system.platform`	string	The entity's operating system distribution
`event.entity.system.platform_family`	string	The entity's operating system family
`event.entity.syst`	strin	The entity's operating system version

`em.platform_version`	g	
`event.entity.user`	string	Sensu RBAC username used by the agent entity

Build event filter expressions with JavaScript execution functions

***COMMERCIAL FEATURE**: Access built-in JavaScript event filter execution functions in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

In addition to Sensu query expressions, Sensu includes several built-in JavaScript functions for event filter execution:

- `sensu.FetchEvent`
- `sensu.CheckStatus`
- `sensu.ListEvents`

Use these functions to query your event stores for other events in the same namespace.

For example, to handle only events for the `server01` entity *and* the `disk` check, use the `sensu.FetchEvent` function in your event filter expressions:

```
"expressions": [
  '(function () { var diskEvent = sensu.FetchEvent("server01", "disk"); if (diskEvent == nil) { return false; } return diskEvent.check.status == 0; })()'
]
```

`sensu.EventStatus`

The `sensu.EventStatus` function takes zero or more checks as arguments. It returns an array of status codes for the events associated with the specified checks.

If you do not specify any checks, the function always returns an empty array.

You can refer to the checks as strings:

```
sensu.EventStatus("database", "disk")
```

If you pass the check names as strings, Sensu assumes that the entities are the same as those in the events being filtered.

You can also refer to the checks in objects that include both the entity and check name. For example:

```
sensu.EventStatus({entity: "server01", check: "disk"}, {entity: "server01", check: "database"})
```

In both cases, if no event matches the specified entities and checks, Sensu will raise an error.

`sensu.FetchEvent`

The `sensu.FetchEvent` function loads the Sensu event that corresponds to the specified entity and check names.

The format is `sensu.FetchEvent(entity, check)`. For example:

```
sensu.FetchEvent("server01", "disk")
```

You can only load events from the same namespace as the event being filtered. The returned object uses the same format as responses for the core/v2/events API.

If an event does not exist for the specified entity and check names, Sensu will raise an error.

`sensu.ListEvents`

The `sensu.ListEvents` function returns an array of all events in the same namespace as the event being filtered.

***NOTE**: If you have many events in the namespace, this function may require a substantial amount of time to return them.*

For example:

```
sensu.ListEvents()
```

The events in the returned array use the same format as responses for the core/v2/events API.

Event filter specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For event filters in this version of Sensu, this attribute should always be `core/v2`.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata	
description	Top-level collection of metadata about the event filter, including `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the filter definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: filter-weekdays-only` `  namespace: default` `  created_by: admin` `  labels:` `  region: us-west-1` `  annotations:` `  slack-channel: "#monitoring"` **JSON** `{` `  "metadata": {` `  "name": "filter-weekdays-only",` `  "namespace": "default",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1"` `  },` `  "annotations": {` `  "slack-channel": "#monitoring"` `  }` `  }` `}`

spec

description	Top-level map that includes the event filter spec attributes.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** ```yaml spec: action: allow expressions: - event.entity.namespace == 'production' runtime_assets: [] ``` **JSON** ```json { "spec": { "action": "allow", "expressions": ["event.entity.namespace == 'production'"], "runtime_assets": [] } } ```

type

description	Top-level attribute that specifies the `sensuctl create` resource type. Event filters should always be type `EventFilter`.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

example	**YML** `type: EventFilter` **JSON** `{ "type": "EventFilter" }`

Metadata attributes

annotations	
description	Non-identifying metadata to include with event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations: managed-by: ops playbook: www.example.url` **JSON** `{ "annotations": { "managed-by": "ops",`

```
    "playbook": "www.example.url"
  }
}
```

created_by

description	Username of the Sensu user who created the filter or last updated the filter. Sensu automatically populates the `created_by` field when the filter is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

labels

description	Custom attributes to include with event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.

required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

YML

example

```yaml
labels:
  environment: development
  region: us-west-2
```

JSON

```json
{
  "labels": {
    "environment": "development",
    "region": "us-west-2"
  }
}
```

name

description	Unique string used to identify the event filter. Filter names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each filter must have a unique name within its namespace.
required	true
type	String

YML

example

```yaml
name: filter-weekdays-only
```

JSON

```
{
```

```
  "name": "filter-weekdays-only"
}
```

namespace	
description	Sensu RBAC namespace that the event filter belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: production` **JSON** `{ "namespace": "production" }`

Spec attributes

action	
description	Action to take with the event if the event filter expressions match. Read Inclusive and exclusive event filters for more information.
required	true
type	String
allowed values	`allow`, `deny`

example	**YML** `action: allow` **JSON** `{ "action": "allow" }`

expressions

description	Event filter expressions to be compared with event data. You can reference event metadata without including the `metadata` scope (for example, `event.entity.namespace`). In filter expressions, place string values inside single or double quotes.
required	true
type	Array
example	**YML** `expressions:` `- event.check.team == 'ops'` `- event.check.annotations["service_priority"] == 1` **JSON** `{ "expressions": [ "event.check.team == 'ops'", "event.check.annotations[\"service_priority\"] == 1" ] }`

runtime_assets	
description	Dynamic runtime assets to apply to the event filter's execution context. JavaScript files in the lib directory of the dynamic runtime asset will be evaluated.
required	false
type	Array of string
default	[]
example	**YML** `runtime_assets:` `- underscore` **JSON** `{ "runtime_assets": [ "underscore" ] }`

Use JavaScript libraries with Sensu filters

You can include JavaScript libraries in their event filter execution context with dynamic runtime assets. For instance, if you package underscore.js into a Sensu asset, you can use functions from the underscore library for filter expressions:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
```

```
  name: deny_if_failure_in_history
spec:
  action: deny
  expressions:
  - _.reduce(event.check.history, function(memo, h) { return (memo || h.status !=
    0); })
  runtime_assets:
  - underscore
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "deny_if_failure_in_history"
  },
  "spec": {
    "action": "deny",
    "expressions": [
      "_.reduce(event.check.history, function(memo, h) { return (memo || h.status !=
0); })"
    ],
    "runtime_assets": ["underscore"]
  }
}
```

Filter for production events

The following event filter allows handling for only events with a custom entity label `"environment": "production"`:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: production_filter
```

```
spec:
  action: allow
  expressions:
  - event.entity.labels['environment'] == 'production'
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "production_filter"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.entity.labels['environment'] == 'production'"
    ]
  }
}
```

Filter for non-production events

The following event filter discards events with a custom entity label `"environment": "production"`, allowing handling only for events without an `environment` label or events with `environment` set to something other than `production`.

***NOTE**: The value for the `action` attribute is `deny`, so this is an exclusive event filter.If evaluation returns false, the event is handled.*

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: not_production
spec:
  action: deny
```

```
  expressions:
  - event.entity.labels['environment'] == 'production'
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "not_production"
  },
  "spec": {
    "action": "deny",
    "expressions": [
      "event.entity.labels['environment'] == 'production'"
    ]
  }
}
```

Filter for state change only

This example demonstrates how to use the `state_change_only` inclusive event filter to reproduce the behavior of a monitoring system that alerts only on state change:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: state_change_only
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "state_change_only"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": []
  }
}
```

Filter for repeated events

In this example, the `filter_interval_60_hourly` event filter will match event data with a check `interval` of `60` seconds *AND* an `occurrences` value of `1` (the first occurrence) *OR* any `occurrences` value that is evenly divisible by 60 via a modulo operator calculation (calculating the remainder after dividing `occurrences` by 60):

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: filter_interval_60_hourly
spec:
  action: allow
  expressions:
  - event.check.interval == 60
  - event.check.occurrences == 1 || event.check.occurrences % 60 == 0
  runtime_assets: []
```

JSON

```
{
```

```
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "filter_interval_60_hourly"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.interval == 60",
      "event.check.occurrences == 1 || event.check.occurrences % 60 == 0"
    ],
    "runtime_assets": []
  }
}
```

This example will apply the same logic as the previous example but for checks with a 30-second `interval`:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: filter_interval_30_hourly
spec:
  action: allow
  expressions:
  - event.check.interval == 30
  - event.check.occurrences == 1 || event.check.occurrences % 120 == 0
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "filter_interval_30_hourly"
  },
```

```
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.interval == 30",
      "event.check.occurrences == 1 || event.check.occurrences % 120 == 0"
    ],
    "runtime_assets": []
  }
}
```

Filter to reduce alert fatigue for keepalive events

This example `keepalive_timeouts` event filter will match event data with an occurrences value of 1 OR any occurrences value that matches 15 minutes via a modulo operator calculation. This limits keepalive timeout event alerts to the first occurrence and every 15 minutes thereafter.

This example uses conditional JavaScript logic to check for an entity-level annotation, `keepalive_alert_minutes`, and if it exists, parses the annotation value as an integer. If the annotation does not exist, the event filter uses 15 minutes for the alert cadence.

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: keepalive_timeouts
spec:
  action: allow
  expressions:
   - is_incident
   - event.check.occurrences == 1 || event.check.occurrences % parseInt( 60 * (
'keepalive_alert_minutes' in event.entity.annotations ?
parseInt(event.entity.annotations.keepalive_alert_minutes): 15) /
event.check.timeout ) == 0
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "keepalive_timeouts"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "is_incident",
      "event.check.occurrences == 1 || event.check.occurrences % parseInt( 60 * (
'keepalive_alert_minutes' in event.entity.annotations ?
parseInt(event.entity.annotations.keepalive_alert_minutes): 15) /
event.check.timeout ) == 0"
    ],
    "runtime_assets": []
  }
}
```

Filter for events during office hours only

This event filter evaluates the event timestamp to determine if the event occurred between 9 AM and 5 PM UTC on a weekday.Remember that `action` is equal to `allow`, so this is an inclusive event filter.I evaluation returns false, the event will not be handled.

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: nine_to_fiver
spec:
  action: allow
  expressions:
  - weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5
  - hour(event.timestamp) >= 9 && hour(event.timestamp) <= 17
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "nine_to_fiver"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5",
      "hour(event.timestamp) >= 9 && hour(event.timestamp) <= 17"
    ],
    "runtime_assets": []
  }
}
```

Add filter expressions that use the `minute` and `second` custom functions for more granular control. For example, if office hours are 8:30 AM to 5:30 PM:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: 830_to_530
spec:
  action: allow
  expressions:
  - weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5
  - hour(event.timestamp) >= 8 && minute(event.timestamp) >= 30
  - hour(event.timestamp) <= 17 && minute(event.timestamp) <= 30
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
```

```
  "api_version": "core/v2",
  "metadata": {
    "name": "830_to_530"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5",
      "hour(event.timestamp) >= 8 && minute(event.timestamp) >= 30",
      "hour(event.timestamp) <= 17 && minute(event.timestamp) <= 30"
    ],
    "runtime_assets": []
  }
}
```

Filter for events not processed within 30 seconds

This event filter evaluates the event timestamp to determine if the event was created more than 30 seconds since the current time. In other words, this filter sets a 30-second time budget for event processing so you can identify and handle events that aren't processed within 30 seconds.

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: budget_30
spec:
  action: allow
  expressions:
  - seconds_since(event.timestamp) > 30
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "budget_30"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "seconds_since(event.timestamp) > 30"
    ],
    "runtime_assets": []
  }
}
```

Disable alerts without a silence

This filter allows you to disable alerts without creating silences.

Add the filter name to the `filters` object for any pipeline whose handler you want to control. To disable alerts, change the filter's `action` attribute value from `allow` to `deny`.

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: emergency_alert_control
spec:
  action: allow
  expressions:
  - event.has_check
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "emergency_alert_control"
  },
```

```
  "spec": {
    "action": "allow",
    "expressions": [
      "event.has_check"
    ]
  }
}
```

Sensu query expressions reference

Sensu query expressions (SQEs) are JavaScript-based expressions that provide additional functionality for using Sensu, like nested parameters and custom functions.

SQEs are defined in event filters, so they act in the context of determining whether a given event should be passed to the handler. SQEs always receive a single event and some information about that event, like `event.timestamp` or `event.check.interval`.

SQEs always return either `true` or `false`. They are evaluated by the Otto JavaScript VM as JavaScript programs.

Syntax quick reference

operator	description
`===`	Identity
`!==`	Nonidentity
`==`	Equality
`!=`	Inequality
`&&`	Logical AND
`||`	Logical OR
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to

Specification

SQEs are valid ECMAScript 5 (JavaScript) expressions that return either `true` or `false`.Other values are not allowed.If an SQE returns a value besides `true` or `false`, an error is recorded in the Sensu backend log and the filter evaluates to `false`.

Custom functions for weekday, hour, minute, and second

Together, the `weekday`, `hour`, `minute`, and `second` custom functions provide granular control of time-based filter expressions, comparable to cron scheduling.

weekday

The custom function `weekday` returns a number that represents the day of the week of a UNIX epoch time.Sunday is `0`.

For example, if an `event.timestamp` equals 1520275913, which is Monday, March 5, 2018 6:51:53 PM UTC, the following SQE returns `false`:

```
weekday(event.timestamp) == 0
```

hour

The custom function `hour` returns the hour of a UNIX epoch time (in UTC and 24-hour time notation).

For example, if an `event.timestamp` equals 1520275913, which is Monday, March 5, 2018 6:51:53 PM UTC, the following SQE returns `true`:

```
hour(event.timestamp) >= 17
```

minute

The custom function `minute` returns the minute of the hour (0 through 59) of a UNIX epoch time in

UTC and 24-hour time notation.

For example, if an `event.timestamp` equals 1520275913, which is Monday, March 5, 2018 6:51:53 PM UTC, the following SQE returns `false`:

```
minute(event.timestamp) <= 30
```

second

The custom function `second` returns the second of the minute (0 through 59) of a UNIX epoch time in UTC and 24-hour time notation.

For example, if an `event.timestamp` equals 1520275913, which is Monday, March 5, 2018 6:51:53 PM UTC, the following SQE returns `true`:

```
second(event.timestamp) >= 30
```

seconds_since custom function

The custom function `seconds_since` returns the number of seconds (using float64) between the current time and an event's timestamp.

For systems with event processing pressure, you can use `seconds_since` to create alerts for events that are not handled within a certain period. For example, the following SQE represents a 30-second time budget for event processing:

```
seconds_since(event.timestamp) > 30
```

sensu.CheckDependencies custom function

Use the `sensu.CheckDependencies` SQE to filter events based on the results of a different check.

The `sensu.CheckDependencies` SQE takes zero or more checks as arguments against the event being filtered. It returns `true` if all the specified checks are passing or `false` if any of the specified checks are failing.

If you do not specify any checks, the `sensu.CheckDependencies` SQE always returns `true`. If no event matches the specified checks, Sensu will raise an error.

You can refer to checks as strings, objects, arrays of strings, and arrays of objects in the `sensu.CheckDependencies` SQE. If you pass the check names as strings, Sensu assumes that the entities are the same as those in the events being filtered. You can also pass entity names and check names in objects to reference checks on specific entities.

String example

In this example, if all checks named `database` or `disk` are passing, the SQE returns `true`:

```
sensu.CheckDependencies("database", "disk")
```

Object example

You can refer to the checks in objects that include both the entity and check name. For example:

```
sensu.CheckDependencies({entity: "server01", check: "disk"}, {entity: "server01",
check: "database"})
```

String and object example

This example mixes string and object references in the same expression. It passes a check name (`disk`) as well as an object that includes entity and check names:

```
sensu.CheckDependencies("disk", {entity: "server01", check: "database"})
```

Array examples

You can use `sensu.CheckDependencies` to evaluate a check that contains an array of elements, which is useful for evaluating arrays parsed from event annotations.

This example references an array of three check names:

```
sensu.CheckDependencies(["port1", "port2", "port3"])
```

This example references an array of objects that each include both an entity and a check name:

```
sensu.CheckDependencies([{entity: "router", check: "port1"}, {entity: "router", check: "port2"}])
```

Examples

Evaluate an event attribute

This SQE returns `true` if the event's entity contains a custom attribute named `namespace` that is equal to `production`:

```
event.entity.namespace == 'production'
```

Evaluate an array

To evaluate an attribute that contains an array of elements, use the `.indexOf` method.For example, this expression returns `true` if an entity includes the subscription `system`:

```
entity.subscriptions.indexOf('system') >= 0
```

Evaluate the day of the week

This expression returns `true` if the event occurred on a weekday:

```
weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5
```

Evaluate office hours

This expression returns `true` if the event occurred between 9 AM and 5 PM UTC:

```
hour(event.timestamp) >= 9 && hour(event.timestamp) <= 17
```

Evaluate labels and annotations

Although you can use annotations to create SQEs, we recommend using labels because labels provide identifying information.

This expression returns `true` if the event's entity includes the label `webserver`:

```
!!event.entity.labels.webserver
```

Likewise, this expression returns `true` if the event's entity includes the annotation `www.company.com`:

```
!!event.entity.annotations['www.company.com']
```

Reduce alert fatigue with event filters

Sensu event filters allow you to filter events destined for one or more event handlers. Filters evaluate their expressions against the observation data in events to determine whether the event should be passed to an event handler.

Use event filters to customize alert policies, improve contact routing, eliminate notification noise from recurring events, and filter events from systems in pre-production environments.

In this guide, you'll learn how to reduce alert fatigue by configuring an event filter named `hourly`. You'll then add the filter to a pipeline workflow that includes a handler named `slack` to prevent alerts from being sent to Slack every minute.

You can take either of two approaches to create the event filter to handle occurrences: use sensuctl or use a filter dynamic runtime asset.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl. In addition, if you don't already have a Slack handler in place, follow Send Slack alerts with handlers to create one before continuing with this guide.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition.

The examples for both approaches in this guide use the `check_cpu` check from Monitor server resources with checks, which includes the subscription `system`. Use sensuctl to add a `system` subscription to one of your entities.

Before you run the following code, replace `<ENTITY_NAME>` with the name of the entity on your system.

***NOTE**: To find your entity name, run `sensuctl entity list`. The `ID` is the name of your entity.*

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `system` and press enter.

Run this command to confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Approach 1: Use sensuctl to create an event filter

First, create an event filter called `hourly` that matches new events (where the event's `occurrences` is equal to `1`) or hourly events (every hour after the first occurrence, calculated with the check's `interval` and the event's `occurrences`).

Events in Sensu Go are handled regardless of check execution status.Even successful check events are passed through the pipeline, so you'll need to add a clause for non-zero status.

```
sensuctl filter create hourly \
--action allow \
--expressions "event.check.occurrences == 1 || event.check.occurrences % (3600 / event.check.interval) == 0"
```

You should receive a confirmation message:

```
Created
```

To view the event filter resource definition, run:

SHELL

```
sensuctl filter info hourly --format yaml
```

SHELL

```
sensuctl filter info hourly --format wrapped-json
```

The event filter definition will be similar to this example:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: hourly
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0
  runtime_assets: null
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "hourly"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0"
    ],
    "runtime_assets": null
  }
}
```

If you want to share and reuse this event filter like code, you can save it to a file and start building a monitoring as code repository.

> ***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Add the event filter to a pipeline

Now that you've created the `hourly` event filter, you can include it in a new pipeline, along with the `slack` handler created in Send Slack alerts with handlers. You'll also include the built-in `is_incident` filter so that only failing events are handled, which will further reduce the number of Slack messages Sensu sends.

> ***NOTE**: If you haven't already created the `slack` handler, follow Send Slack alerts with handlers before continuing with this step.*

To create a new pipeline that includes the `hourly` and `is_incident` event filters as well as the `slack` handler, run:

SHELL

```
echo '---
type: Pipeline
api_version: core/v2
metadata:
  name: reduce_alerts
spec:
  workflows:
  - name: slack_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: hourly
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2' | sensuctl create
```

SHELL

```
echo '{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "reduce_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "slack_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "hourly",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}' | sensuctl create
```

Assign the pipeline to a check

To use the `reduce_alerts` pipeline, list it in a check definition's pipelines array. This example uses the `check_cpu` check created in Monitor server resources with checks). All the observability events that

the check produces will be processed according to the pipeline's workflows.

Assign your `reduce_alerts` pipeline to the `check_cpu` check to receive Slack alerts when the CPU usage of your system reaches the specific thresholds set in the check command.

To open the check definition in your text editor, run:

```
sensuctl edit check check_cpu
```

Replace the `pipelines: []` line with the following array and save the updated check definition:

```
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: reduce_alerts
```

You should see a response to confirm the update:

```
Updated /api/core/v2/namespaces/default/checks/check_cpu
```

To view the updated `check_cpu` resource definition, run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The updated check definition will be similar to this example:

YML

```
---
```

```yaml
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: null
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: reduce_alerts
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
```

```
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "name": "reduce_alerts",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```


The check will now send events to the `reduce_alerts` pipeline.Skip to Confirm the event filter to lear how to verify that the filter is working.

Approach 2: Use an event filter dynamic runtime asset

If you're not already familiar with dynamic runtime assets, read Use assets to install plugins.This will help you understand what dynamic runtime assets are and how they are used in Sensu.

In this approach, the first step is to obtain an event filter dynamic runtime asset that will allow you to replicate the behavior of the `hourly` event filter created in Approach 1 via `sensuctl`.

Use `sensuctl asset add` to register the sensu/sensu-go-fatigue-check-filter dynamic runtime asset:

```
sensuctl asset add sensu/sensu-go-fatigue-check-filter:0.8.1 -r fatigue-filter
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `fatigue-filter`.

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-go-fatigue-check-filter:0.8.1
added asset: sensu/sensu-go-fatigue-check-filter:0.8.1

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["fatigue-filter"].
```

You can also download the asset directly from Bonsai, the Sensu asset hub.

> ***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

You've registered the dynamic runtime asset, but you still need to create the filter.

Create a file named `sensu-fatigue-check-filter.yml` or `sensu-fatigue-check-filter.json` in your Sensu installation to store the event filter definition.Copy this this filter definition into the file and save it:

YML

```
---
type: EventFilter
api_version: core/v2
```

```
metadata:
  name: fatigue_check
spec:
  action: allow
  expressions:
  - fatigue_check(event)
  runtime_assets:
  - fatigue-filter
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "fatigue_check"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "fatigue_check(event)"
    ],
    "runtime_assets": [
      "fatigue-filter"
    ]
  }
}
```

Then, use sensuctl to create a filter named `fatigue_check` from the file:

SHELL

```
sensuctl create -f sensu-fatigue-check-filter.yml
```

SHELL

```
sensuctl create -f sensu-fatigue-check-filter.json
```

Now that you've added the dynamic runtime asset and created the event filter definition and pipeline, you can create the check annotations you need for the dynamic runtime asset to work properly.

Update a check for filter dynamic runtime asset use

Next, you'll need to make some additions to any checks you want to use the `fatigue_check` filter with. This example uses the `check_cpu` check created in Monitor server resources with checks). All the observability events that the check produces will be processed according to the pipeline's workflows.

Assign your `reduce_alerts` pipeline to the `check_cpu` check to receive Slack alerts when the CPU usage of your system reaches the specific thresholds set in the check command.

To open the check definition in your text editor, run:

```
sensuctl edit check check_cpu
```

In the check definition, update the `pipelines: []` line with the following array:

SHELL

```
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: cpu_check_alerts
```

SHELL

```
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "cpu_check_alerts"
    }
  ]
```

Add the following annotations in the check metadata:

SHELL

```
  annotations:
    fatigue_check/occurrences: '1'
    fatigue_check/interval: '3600'
    fatigue_check/allow_resolution: 'false'
```

SHELL

```
  "annotations": {
    "fatigue_check/occurrences": "1",
    "fatigue_check/interval": "3600",
    "fatigue_check/allow_resolution": "false"
  }
```

After you add the pipeline array and annotations, save the updated check definition. To confirm your updates, run this command to retrieve the check definition:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The check definition should be similar to this example:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: cpu-check
  annotations:
    fatigue_check/occurrences: '1'
    fatigue_check/interval: '3600'
    fatigue_check/allow_resolution: 'false'
```

```
spec:
  command: check-cpu -w 75 c 95
  env_vars: null
  handlers: null
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ''
  output_metric_handlers: null
  output_metric_tags: null
  pipelines:
  - api_version: core/v2
    name: reduce_alerts
    type: Pipeline
  proxy_entity_name: ''
  publish: true
  round_robin: false
  runtime_assets:
  - check_cpu_usage
  stdin: false
  subdue:
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "cpu-check",
    "annotations": {
      "fatigue_check/occurrences": "1",
      "fatigue_check/interval": "3600",
      "fatigue_check/allow_resolution": "false"
    }
  },
  "spec": {
    "command": "check-cpu -w 75 c 95",
    "env_vars": null,
```

```
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "output_metric_tags": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "name": "reduce_alerts",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check_cpu_usage"
    ],
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```


The annotations are required for the filter dynamic runtime asset to work the same way as the interactively created event filter.Specifically, the annotations in this check definition are doing several things:

1. `fatigue_check/occurrences` : Tells the event filter on which occurrence to send the event for further processing
2. `fatigue_check/interval` : Tells the event filter the interval at which to allow additional events to be processed (in seconds)
3. `fatigue_check/allow_resolution` : Determines whether to pass a `resolve` event through to the filter

For more information about configuring these values, read the Sensu Go Fatigue Check Filter README.Next, you'll add the newly minted event filter and an existing handler to a pipeline.

Add the event filter to a pipeline

Now that you've created the `fatigue_check` event filter, you can add it to a pipeline along with the `slack` handler created in Send Slack alerts with handlers. You'll also add the built-in `is_incident` filter so that only failing events are handled, which will further reduce the number of Slack messages Sensu sends.

> ***NOTE**: If you haven't already created the `slack` handler, follow Send Slack alerts with handlers before continuing with this step.*

SHELL

```
echo '---
type: Pipeline
api_version: core/v2
metadata:
  name: reduce_alerts
spec:
  workflows:
  - name: slack_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: fatigue_check
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2' | sensuctl create
```

SHELL

```
echo '{
  "type": "Pipeline",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "reduce_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "slack_alerts",
        "filters": [
          {
            "name": "fatigue_check",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "hourly",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}' | sensuctl create
```

Confirm the event filter

Instead of waiting to receive a Slack alert, you can verify the proper behavior of these event filters with `sensu-backend` logs. The default location of these logs varies based on your platform. Read Troubleshoot Sensu for details about the log location.

Whenever an event is being handled, two log entries are added:

```
"handler":"slack","level":"debug","msg":"sending event to handler"
"msg":"pipelined executed event pipe handler","output":"","status":0
```

However, if the event is being discarded by the event filter, a log entry with the message `event filtered` will appear instead.

Next steps

Now that you know how to add event filters to pipelines and use a dynamic runtime asset to help reduce alert fatigue, read the filters reference for in-depth information about event filters.

Route alerts with event filters

Every alert has an ideal first responder: a team or person who knows how to triage and address the issue. Sensu contact routing lets you alert the right people using their preferred contact methods and reduce mean time to response and recovery.

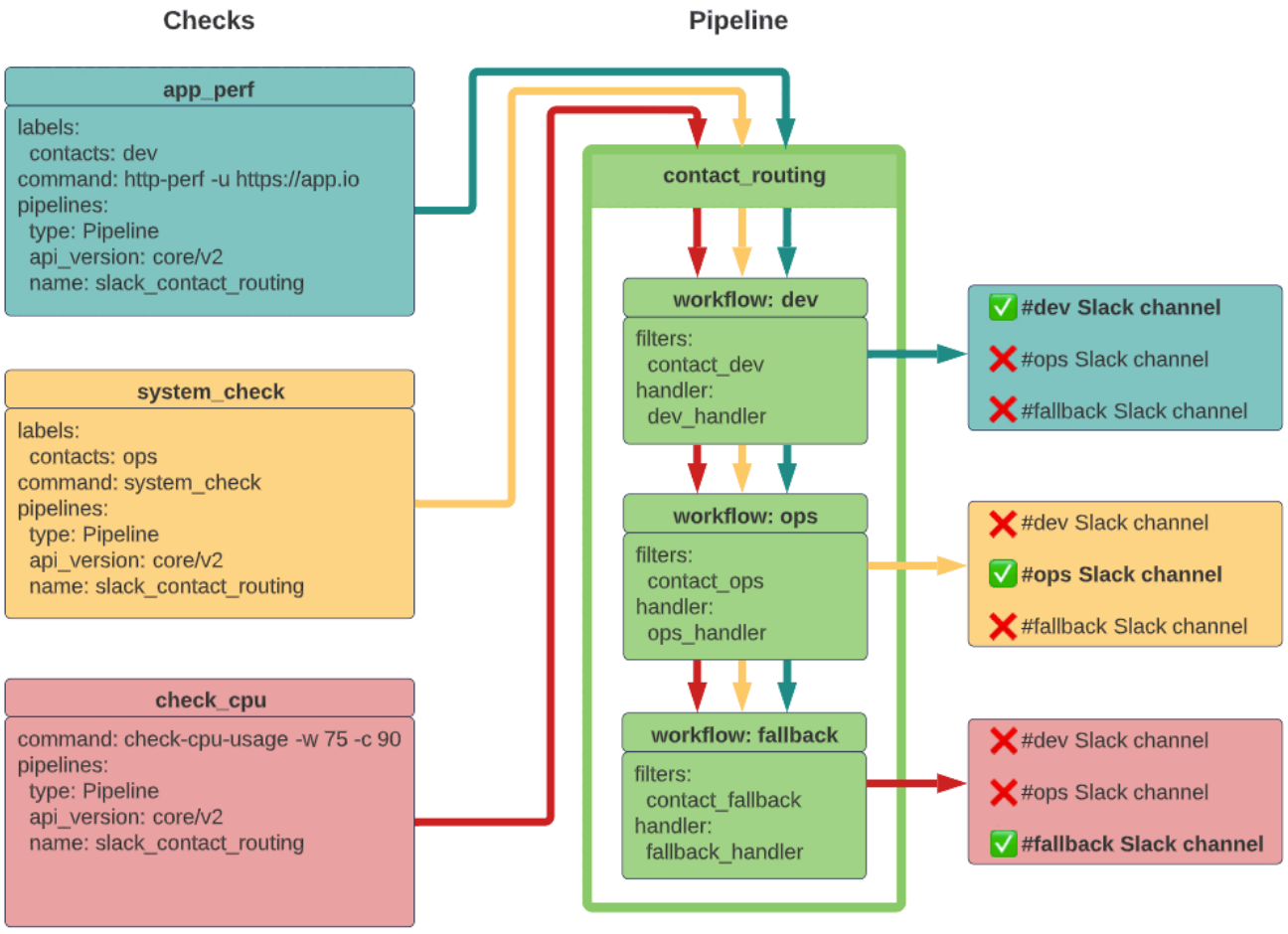
In this guide, you'll set up alerts for two teams (dev and ops) with separate Slack channels. Each team wants to be alerted only for the things they care about, using their team's Slack channel. There's also a fallback option for alerts that should not be routed to either the dev or ops team. To achieve this, you'll use a pipeline resource with three workflows, one for each contact option.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl. You will also need cURL and a Slack webhook URL and three different Slack channels to receive test alerts (one for each team).

Routing alerts requires three types of Sensu resources:

- **Handlers** to store contact preferences for the dev and ops teams, plus a fallback option
- **Event filters** to match contact labels to the right handler
- A **pipeline** to organize the event filters and handlers into workflows that route alerts to the right contacts

Here's a quick overview of the configuration to set up contact routing with a pipeline. Two of the check definitions include a `contacts` label, which allows the pipeline to route alerts to the correct Slack channel based each workflow's event filter and handler.



Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition.

This guide uses an example check that includes the subscription `system`. Use sensuctl to add a `system` subscription to one of your entities.

Before you run the following code, replace `<ENTITY_NAME>` with the name of the entity on your system.

***NOTE**: To find an entity's name, run `sensuctl entity list`. The `ID` is the name of the entity.*

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `system` and press enter.

Run this command to confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register dynamic runtime assets

Contact routing is powered by the sensu/sensu-go-has-contact-filter dynamic runtime asset. To add the asset to Sensu, use sensuctl asset add:

```
sensuctl asset add sensu/sensu-go-has-contact-filter:0.3.0 -r contact-filter
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-go-has-contact-filter:0.3.0
added asset: sensu/sensu-go-has-contact-filter:0.3.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["contact-filter"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `contact-filter`.

Next, add the sensu/sensu-slack-handler dynamic runtime asset to Sensu with sensuctl:

```
sensuctl asset add sensu/sensu-slack-handler:1.5.0 -r sensu-slack-handler
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-slack-handler:1.5.0 -r sensu-slack-handler
added asset: sensu/sensu-slack-handler:1.5.0

You have successfully added the Sensu asset resource, but the asset will not get downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the appropriate
resource, populate the "runtime_assets" field with ["sensu-slack-handler"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `sensu-slack-handler`.

Run `sensuctl asset list` to confirm that the dynamic runtime assets are ready to use. The response will confirm the available assets:

```
     Name                    URL                     Hash
_____________________ _____________________________________________________________ _________

 contact-filter       //assets.bonsai.sensu.io/.../sensu-go-has-contact-filter_0.3.0.tar.gz       d35c6c4
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_windows_amd64.tar.gz  53359fa
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_darwin_386.tar.gz     e2d7d0d
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_linux_armv7.tar.gz    362fe51
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_linux_arm64.tar.gz    b492ae2
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_darwin_amd64.tar.gz   88bbdca
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_linux_386.tar.gz      d9040ae
 sensu-slack-handler  //assets.bonsai.sensu.io/.../sensu-slack-handler_1.0.3_linux_amd64.tar.gz    6872086
```

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read the asset reference for more information about dynamic runtime asset builds.*

Create contact filters

The Bonsai documentation explains that the sensu/sensu-go-has-contact-filter dynamic runtime asset supports two functions:

- `has_contact`, which takes the Sensu event and the contact name as arguments
- `no_contact`, which is available as a fallback in the absence of contact labels and takes only the event as an argument

You'll use these functions to create event filters that represent the three actions that the Sensu Slack handler can take on an event: contact the ops team, contact the dev team, and contact the fallback option.

event filter name	expression	description
`contact_ops`	`has_contact(event, "ops")`	Allow events with the entity or check label `contacts: ops`
`contact_dev`	`has_contact(event, "dev")`	Allow events with the entity or check label `contacts: dev`
`contact_fallback`	`no_contacts(event)`	Allow events without an entity or check `contacts` label

Use sensuctl to create the three event filters:

SHELL

```
echo '---
type: EventFilter
api_version: core/v2
metadata:
  name: contact_ops
spec:
  action: allow
  runtime_assets:
    - contact-filter
  expressions:
    - has_contact(event, "ops")
---
type: EventFilter
api_version: core/v2
metadata:
```

```
  name: contact_dev
spec:
  action: allow
  runtime_assets:
    - contact-filter
  expressions:
    - has_contact(event, "dev")
---
type: EventFilter
api_version: core/v2
metadata:
  name: contact_fallback
spec:
  action: allow
  runtime_assets:
    - contact-filter
  expressions:
    - no_contacts(event)' | sensuctl create
```

SHELL

```
echo '{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "contact_ops"
  },
  "spec": {
    "action": "allow",
    "runtime_assets": [
      "contact-filter"
    ],
    "expressions": [
      "has_contact(event, \"ops\")"
    ]
  }
}
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "contact_dev"
  },
  "spec": {
    "action": "allow",
    "runtime_assets": [
      "contact-filter"
    ],
    "expressions": [
      "has_contact(event, \"dev\")"
    ]
  }
}
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "contact_fallback"
  },
  "spec": {
    "action": "allow",
    "runtime_assets": [
      "contact-filter"
    ],
    "expressions": [
      "no_contacts(event)"
    ]
  }
}' | sensuctl create
```

You can also save these event filter resource definitions to a file named `filters.yml` or `filters.json` in your Sensu installation. When you're ready to manage your observability configurations the same way you do any other code, your `filters.yml` or `filters.json` file can become a part of your monitoring as code repository.

Use sensuctl to confirm that the event filters were added:

```
sensuctl filter list
```

The response should list the new `contact_ops`, `contact_dev`, and `contact_fallback` event

filters:

```
     Name        Action         Expressions
──────────────────────── ──────── ──────────────────────────────────
contact_dev         allow    (has_contact(event, "dev"))
contact_fallback    allow    (no_contacts(event))
contact_ops         allow    (has_contact(event, "ops"))
```

Create a handler for each contact

With your contact filters in place, you can create a handler for each contact: ops, dev, and fallback. In each handler definition, you will specify:

- A unique name: `ops_handler`, `dev_handler`, or `fallback_handler`
- A customized command with the contact's preferred Slack channel
- An environment variable that contains your Slack webhook URL
- The `sensu-slack-handler` dynamic runtime asset

Before you run the following code to create the handlers with sensuctl, make these changes:

- Replace `<ALERT_OPS>`, `<ALERT_DEV>`, and `<ALERT_ALL>` with the names of the channels you want to use to receive alerts in your Slack instance.
- Replace `<SLACK_WEBHOOK_URL>` with your Slack webhook URL.

After you update the code to use your preferred Slack channels and webhook URL, run:

SHELL

```
echo '---
type: Handler
api_version: core/v2
metadata:
  name: ops_handler
spec:
  command: sensu-slack-handler --channel "#<ALERT_OPS>"
  env_vars:
    - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx
```

```
  handlers: null
  runtime_assets:
    - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe
---
type: Handler
api_version: core/v2
metadata:
  name: dev_handler
spec:
  command: sensu-slack-handler --channel "#<ALERT_DEV>"
  env_vars:
    - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx
  handlers: null
  runtime_assets:
    - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe
---
type: Handler
api_version: core/v2
metadata:
  name: fallback_handler
spec:
  command: sensu-slack-handler --channel "#<ALERT_ALL>"
  env_vars:
    - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx
  handlers: null
  runtime_assets:
    - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe' | sensuctl create
```

SHELL

```
echo '{
  "type": "Handler",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "ops_handler"
  },
  "spec": {
    "command": "sensu-slack-handler --channel \"#<ALERT_OPS>\"",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx"
    ],
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "dev_handler"
  },
  "spec": {
    "command": "sensu-slack-handler --channel \"#<ALERT_DEV>\"",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx"
    ],
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "fallback_handler"
```

```
  },
  "spec": {
    "command": "sensu-slack-handler --channel \"#<ALERT_ALL>\"",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx"
    ],
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}' | sensuctl create
```

Just like the event filters, you can save these handlers to a YAML or JSON file to create a handlers configuration file if you're implementing monitoring as code.

Use sensuctl to confirm that the handlers were added:

```
sensuctl handler list
```

The response should list the new `dev_handler`, `ops_handler`, and `fallback_handler` handlers:

```
      Name        Type   Timeout   Filters   Mutator                  Execute                                   Environment
Variables                     Assets
────────────────── ────── ───────── ───────── ─────────
─────────────────────────────────────────────────────
───────────────────────────────────────────────────────── ────────────────────
 dev_handler       pipe        0                    RUN:    sensu-slack-handler --channel "#<ALERT_DEV>"
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx   sensu-slack-handler
 fallback_handler   pipe        0                    RUN:    sensu-slack-handler --channel "#<ALERT_ALL>"
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx   sensu-slack-handler
 ops_handler       pipe        0                    RUN:    sensu-slack-handler --channel "#<ALERT_OPS>"
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/xxxxxxxxx   sensu-slack-handler
```

Create a pipeline

Create a pipeline with a three workflows: one for each contact group.

Each workflow includes the contact event filter and the corresponding handler for one contact group. All of the workflows also include the built-in is_incident event filter to reduce noise.

SHELL

```
echo '---
type: Pipeline
api_version: core/v2
metadata:
  name: slack_contact_routing
spec:
  workflows:
  - name: dev
    filters:
    - name: contact_dev
      type: EventFilter
      api_version: core/v2
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: dev_handler
      type: Handler
      api_version: core/v2
  - name: ops
    filters:
    - name: contact_ops
      type: EventFilter
      api_version: core/v2
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: ops_handler
      type: Handler
      api_version: core/v2
  - name: fallback
```

```
    filters:
    - name: contact_fallback
      type: EventFilter
      api_version: core/v2
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: fallback_handler
      type: Handler
      api_version: core/v2' | sensuctl create
```

SHELL

```
echo '{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack_contact_routing"
  },
  "spec": {
    "workflows": [
      {
        "name": "dev",
        "filters": [
          {
            "name": "contact_dev",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "dev_handler",
          "type": "Handler",
          "api_version": "core/v2"
        }
```

```
    },
    {
      "name": "ops",
      "filters": [
        {
          "name": "contact_ops",
          "type": "EventFilter",
          "api_version": "core/v2"
        },
        {
          "name": "is_incident",
          "type": "EventFilter",
          "api_version": "core/v2"
        }
      ],
      "handler": {
        "name": "ops_handler",
        "type": "Handler",
        "api_version": "core/v2"
      }
    },
    {
      "name": "fallback",
      "filters": [
        {
          "name": "contact_fallback",
          "type": "EventFilter",
          "api_version": "core/v2"
        },
        {
          "name": "is_incident",
          "type": "EventFilter",
          "api_version": "core/v2"
        }
      ],
      "handler": {
        "name": "fallback_handler",
        "type": "Handler",
        "api_version": "core/v2"
      }
    }
  ]
```

```
  }
}' | sensuctl create
```

With your pipeline in place, you can send ad hoc events to test your configuration and make sure the right contact groups receive the right alerts in Slack.

Send events to test your configuration

Use the agent API to create ad hoc events and send them to your Slack pipeline.

First, create an event without a `contacts` label. You may need to modify the URL with your Sensu agent address.

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "example-check-fallback"
    },
    "status": 1,
    "output": "You should receive this example event in the Slack channel specified
by your fallback handler."
  },
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "contact_routing"
    }
  ]
}' \
http://127.0.0.1:3031/events
```

Since this event doesn't include a `contacts` label, you should also receive an alert in the Slack channel specified in your `fallback_handler` handler. Behind the scenes, Sensu uses the `contact_fallback` filter to match the event to the `fallback_handler` handler.

Now, create an event with a `contacts` label:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "example-check-dev",
      "labels": {
        "contacts": "dev"
      }
    },
    "status": 1,
    "output": "You should receive this example event in the Slack channel specified
by your dev handler."
  },
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "contact_routing"
    }
  ]
}' \
http://127.0.0.1:3031/events
```

Because this event contains the `contacts: dev` label, you should receive an alert in the Slack channel specified by the `dev_handler` handler.

Resolve the events by sending the same API requests with `status` set to `0`.

Manage contact labels in checks and entities

To assign a check's alerts to a contact, you can add the `contacts` labels to checks or entities.

Route contacts with checks

To test contact routing with check-generated events, update the `check_cpu` check created in Monitor server resources to include the `ops` and `dev` contacts and the `slack_contact_routing` pipeline.

Use sensuctl to open the check in a text editor:

```
sensuctl edit check check_cpu
```

Edit the check metadata to add the following labels:

YML

```
labels:
  contacts: dev, ops
```

JSON

```
{
  "labels": {
    "contacts": "dev, ops"
  }
}
```

Update the pipelines array to add `slack_contact_routing`:

YML

```
pipelines:
  - type: Pipeline
    api_version: core/v2
    name: slack_contact_routing
```

JSON

```
{
  "pipelines": {
    "type": "Pipeline",
    "api_version": "core/v2",
    "name": "slack_contact_routing"
```

```
  }
}
```

Save and close the updated check definition. A response will confirm the check was updated. For example:

```
Updated /api/core/v2/namespaces/default/checks/check_cpu
```

To view the updated resource definition for `check_cpu` and confirm that it includes the `contacts` labels and `slack_contact_routing` pipeline, run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The sensuctl response will include the updated `check_cpu` resource definition in the specified format:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  labels:
    contacts: dev, ops
  name: check_cpu
  namespace: default
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: []
```

```
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: slack_contact_routing
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

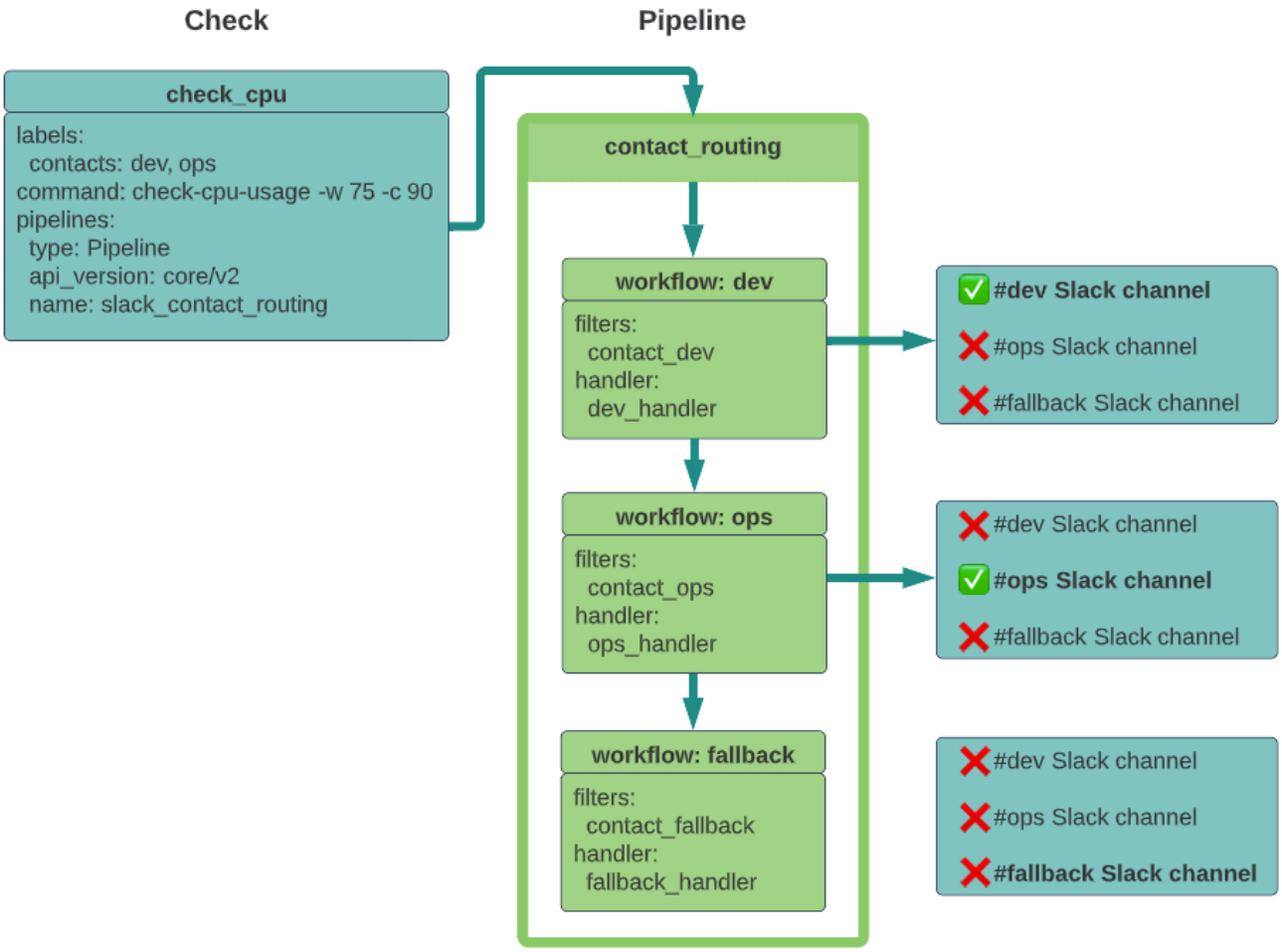
JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu",
    "namespace": "default",
    "labels": {
      "contacts": "dev, ops"
    },
    "created_by": "admin"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
```

```
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "name": "slack_contact_routing",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

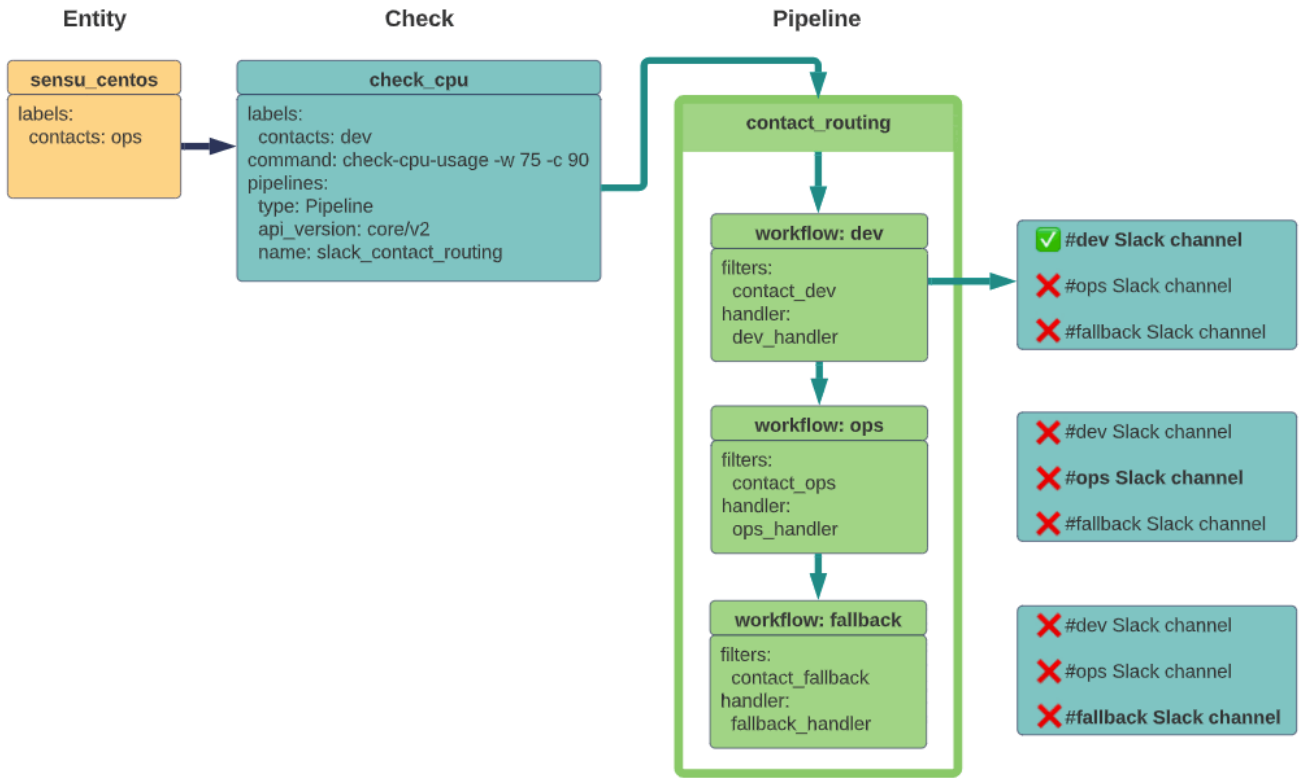
Now when the `check_cpu` check generates an event, Sensu will filter the event according to the `contact_dev` and `contact_ops` event filters and send alerts to the #dev and #ops Slack channels:



Entities

You can specify contacts in entity labels instead of in check labels. The check definition should still include the pipeline. For more information about managing entity labels, read the entities reference.

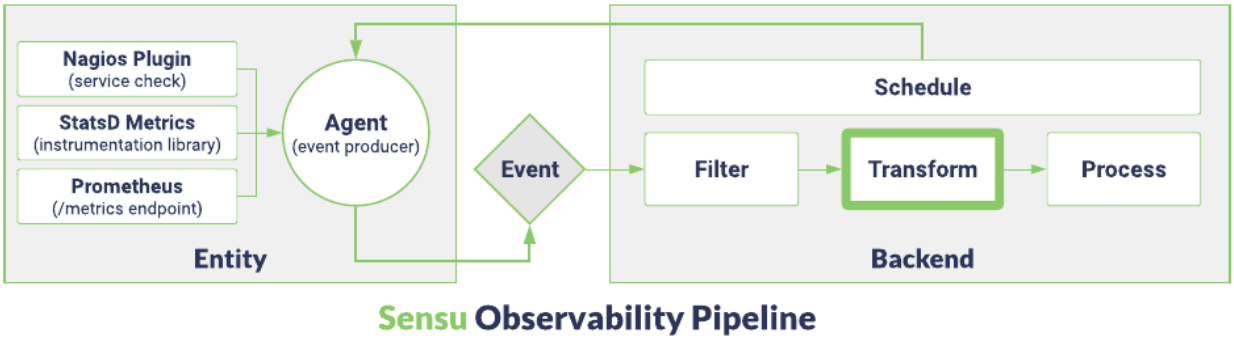
If contact labels are present in both the check and entity, the check contacts override the entity contacts. In this example, the `dev` label in the check configuration overrides the `ops` label in the agent definition, resulting in an alert sent to #dev but not to #ops or #fallback:



Next steps

Now that you've set up contact routing for two example teams, you can create additional filters, handlers, and labels to represent your team's contacts. Learn how to use Sensu to Reduce alert fatigue.

Transform your observation data



Sensu Observability Pipeline

Next **or click any element in the pipeline to jump to it.**

In the transform stage, Sensu executes mutators.

The transform stage of the Sensu observability pipeline executes any mutators you have specified in your pipeline configuration to transform your observability data so other technologies can consume it. For example, if you're sending metrics to Graphite using a TCP handler, Graphite expects data that follows the Graphite plaintext protocol. You can use Sensu's built-in only_check_output mutator to transform the data into the format Graphite can accept.

Here's how transform stage of the pipeline works: first, the Sensu backend receives an event and executes the filter stage of the observability pipeline. If the event data meets the conditions, triggers, or thresholds you specified in your event filters, Sensu checks the pipeline for a mutator. If the pipeline includes a mutator, the Sensu backend executes the mutator.

There are two types of mutators: pipe and JavaScript.

Pipe mutators

Pipe mutator definitions include executable commands that will be executed on a Sensu backend. Pipe mutators produce an exit status code to indicate state.

- If the mutator executes successfully (that is, returns an exit status code of `0`), Sensu applies the mutator to transform the event data, returns the transformed event data to the handler specified in the pipeline, and executes the handler.

- If the mutator fails to execute (that is, returns a non-zero exit status code or fails to complete within its configured timeout), Sensu logs an error and does not execute the handler specified in the pipeline.

This example pipe mutator resource definition uses the Sensu Check Status Metric Mutator dynamic runtime asset:

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: sensu-check-status-metric-mutator
spec:
  command: sensu-check-status-metric-mutator
  runtime_assets:
  - nixwiz/sensu-check-status-metric-mutator
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-check-status-metric-mutator"
  },
  "spec": {
    "command": "sensu-check-status-metric-mutator",
    "runtime_assets": [
      "nixwiz/sensu-check-status-metric-mutator"
    ]
  }
}
```

Most pipe mutator commands are provided by Sensu plugins, which you can deploy with dynamic runtime assets.Use Bonsai, the Sensu asset hub, to discover, download, and share dynamic runtime assets for Sensu pipe mutators.Read Use assets to install plugins to get started.

JavaScript mutators

JavaScript mutators allow you to write your own evaluation expressions and do not require an executable command attribute.Each Sensu JavaScript mutator definition includes the eval attribute, whose value must be an ECMAScript 5 expression.

This example uses a JavaScript mutator to remove event attributes (in this case, the check name and entity `app_id` label):

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: remove_checkname_entitylabel
spec:
  eval: >-
    data = JSON.parse(JSON.stringify(event)); delete data.check.metadata.name;
    delete data.entity.metadata.labels.app_id; return JSON.stringify(data)
  type: javascript
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "remove_checkname_entitylabel"
  },
  "spec": {
    "eval": "data = JSON.parse(JSON.stringify(event)); delete
data.check.metadata.name; delete data.entity.metadata.labels.app_id; return
JSON.stringify(data)",
    "type": "javascript"
  }
}
```

Mutators reference

Sensu executes mutators during the **transform** stage of the observability pipeline.

Pipelines can specify a mutator to execute and transform observability event data before any handlers are applied. When the Sensu backend processes an event, it checks the pipeline for the presence of a mutator and executes that mutator before executing the handler.

Mutators accept input/data via stdin and can parse JSON event data. They output JSON data (modified event data) to stdout or stderr.

There are two types of mutators: pipe and JavaScript.

Pipe mutator examples

This example shows a pipe mutator resource definition with the minimum required attributes:

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: mutator_minimum
spec:
  command: example_mutator.go
  type: pipe
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "mutator_minimum"
  },
```

```
  "spec": {
    "command": "example_mutator.go",
    "type": "pipe"
  }
}
```

The following mutator definition uses an imaginary Sensu plugin, `example_mutator.go`, to modify event data prior to handling the event:

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: example-mutator
spec:
  command: example_mutator.go
  eval: ""
  env_vars: []
  runtime_assets:
  - example-mutator-asset
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "example-mutator"
  },
  "spec": {
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": [
      "example-mutator-asset"
```

```
    ],
    "secrets": null,
    "type": "pipe",
    "eval": ""
  }
}
```

JavaScript mutator example

JavaScript mutators use the eval attribute instead of the command attribute. The eval value must be an ECMAScript 5 (JavaScript) expression.

This example uses a JavaScript mutator to remove event attributes that are not required — in this case, the check name and entity `app_id` label:

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: remove_checkname_entitylabel
spec:
  eval: >-
    data = JSON.parse(JSON.stringify(event)); delete data.check.metadata.name;
    delete data.entity.metadata.labels.app_id; return JSON.stringify(data)
  type: javascript
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "remove_checkname_entitylabel"
  },
  "spec": {
    "eval": "data = JSON.parse(JSON.stringify(event)); delete
data.check.metadata.name; delete data.entity.metadata.labels.app_id; return
```

```
JSON.stringify(data)",
    "type": "javascript"
  }
}
```

You can also use JavaScript mutators to do things like add new attributes and combine existing attributes into a single new attribute.

Pipe mutators

Pipe mutators produce an exit status code to indicate state. A code of `0` indicates OK status. If the mutator executes successfully (returns an exit status code of `0`), the modified event data return to the pipeline and the handler is executed.

Exit codes other than `0` indicate failure. If the mutator fails to execute (returns a non-zero exit status code or fails to complete within its configured timeout), an error is logged and the handler will not execute.

Pipe mutator commands

Each Sensu mutator definition defines a command to be executed. Mutator commands are executable commands that will be executed on a Sensu backend, run as the `sensu` user. Most mutator commands are provided by Sensu plugins.

Sensu mutator `command` attributes may include command line arguments for controlling the behavior of the `command` executable. Many Sensu mutator plugins provide support for command line arguments for reusability.

All mutator commands are executed by a Sensu backend as the `sensu` user. Commands must be executable files that are discoverable on the Sensu backend system (installed in a system `$PATH` directory).

***NOTE**: By default, Sensu installer packages will modify the system `$PATH` for the Sensu processes to include `/etc/sensu/plugins`. As a result, executable scripts (like plugins) located in `/etc/sensu/plugins` will be valid commands. This allows `command` attributes to use "relative paths" for Sensu plugin commands (for example, `"command": "check-http.go -u https://sensuapp.org"`).*

JavaScript mutators

Mutators that use JavaScript are an efficient alternative to pipe mutators, which fork a process on each invocation. JavaScript mutators are evaluated by the Otto JavaScript VM as JavaScript programs, which enables greater mutator throughput at scale.

JavaScript mutators do not require you to return any value — you can mutate the events that are passed to the mutator instead. However, if you do return a value with a JavaScript mutator, it must be a string. If a JavaScript mutator returns a non-string value (an array, object, integer, or Boolean), an error is recorded in the Sensu backend log.

JavaScript mutators can use dynamic runtime assets as long as they are valid JavaScript assets.

Secrets are not available to JavaScript mutators. JavaScript mutators cannot look up events from the event store.

JavaScript mutator eval attribute

Each Sensu JavaScript mutator definition includes the eval attribute, whose value must be an ECMAScript 5 (JavaScript) expression. JavaScript mutators do not use the command attribute.

All mutator eval expressions are executed by a Sensu backend as the `sensu` user.

JavaScript mutator eval expressions can use the environment variables listed in the env_vars attribute. For JavaScript mutators, you can define environment variables and list the names of any environment variables that are available in your environment in the env_vars attribute.

Built-in mutators

Sensu includes built-in mutators to help you customize event pipelines for metrics and alerts.

Built-in mutator: only_check_output

To process an event, some handlers require only the check output, not the entire event definition. For example, when sending metrics to Graphite using a TCP handler, Graphite expects data that follows the Graphite plaintext protocol. By using the built-in `only_check_output` mutator, Sensu reduces the

event to only the check output so Graphite can accept it.

To use only check output, include the `only_check_output` mutator in the pipeline `mutator` array:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: graphite_pipeline
spec:
  workflows:
  - name: graphite_check_output
    filters:
    - name: has_metrics
      type: EventFilter
      api_version: core/v2
    mutator:
      name: only_check_output
      type: Mutator
      api_version: core/v2
    handler:
      name: graphite
      type: Handler
      api_version: core/v2
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "graphite_pipeline"
  },
  "spec": {
    "workflows": [
      {
        "name": "graphite_check_output",
        "filters": [
          {
            "name": "has_metrics",
```

```
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "only_check_output",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "graphite",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

Mutator specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For mutators in this version of Sensu, the `api_version` should always be `core/v2`.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON**

```
{
  "api_version": "core/v2"
}
```

metadata	
description	Top-level collection of metadata about the mutator that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the mutator definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Review the metadata attributes reference for details.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: example-mutator` `  namespace: default` `  created_by: admin` `  labels:` `  region: us-west-1` `  annotations:` `  slack-channel: "#monitoring"` **JSON** `{` `  "metadata": {` `  "name": "example-mutator",` `  "namespace": "default",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1"` `  },`

```
    "annotations": {
      "slack-channel": "#monitoring"
    }
  }
}
```

spec	
description	Top-level map that includes the mutator spec attributes.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  command: example_mutator.go` `  timeout: 0` `  env_vars: []` `  runtime_assets: []` `  secrets: null` `  type: pipe` **JSON** `{` `  "spec": {` `  "command": "example_mutator.go",` `  "timeout": 0,` `  "env_vars": [],` `  "runtime_assets": [],` `  "secrets": null,` `  "type": "pipe"` `  }` `}`

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Mutators should always be type `Mutator`.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: Mutator` **JSON** `{ "type": "Mutator" }`

Metadata attributes

annotations	
description	Non-identifying metadata to include with event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null` **YML**

example

```
annotations:
  managed-by: ops
  playbook: www.example.url
```

JSON

```
{
  "annotations": {
    "managed-by": "ops",
    "playbook": "www.example.url"
  }
}
```

created_by

description	Username of the Sensu user who created the mutator or last updated the mutator. Sensu automatically populates the `created_by` field when the mutator is created or updated.
required	false
type	String

YML

example

```
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

name

description	Unique string used to identify the mutator. Mutator names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each mutator must have a unique name within its namespace.

required	true
type	String
example	YML ```yaml name: example-mutator ``` JSON ```json { "name": "example-mutator" } ```

namespace

description	Sensu RBAC namespace that the mutator belongs to.
required	false
type	String
default	`default`
example	YML ```yaml namespace: production ``` JSON ```json { "namespace": "production" } ```

Spec attributes

command

description	Mutator command to be executed by the Sensu backend. ***NOTE**: JavaScript mutators require the eval attribute instead of the command attribute.*
required	true, for pipe mutators
type	String
example	**YML** `command: /etc/sensu/plugins/mutated.go` **JSON** `{ "command": "/etc/sensu/plugins/mutated.go" }`

env_vars

description	Array of environment variables to use with command or eval expression execution.
required	false
type	Array
example	**YML** `env_vars:` `- APP_VERSION=2.5.0` **JSON** `{`

```
  "env_vars": [
    "APP_VERSION=2.5.0"
  ]
}
```

As of Sensu Go 6.5.2, for JavaScript mutators, you can list any environment variables that are available in your environment in addition to defining environment variables:

YML

```
env_vars:
- APP_VERSION=2.5.0
- SHELL
```

JSON

```
{
  "env_vars": [
    "APP_VERSION=2.5.0",
    "SHELL"
  ]
}
```

eval

description	ECMAScript 5 (JavaScript) expression to be executed by the Sensu backend. ***NOTE**: Pipe mutators require the command attribute instead of the eval attribute.*
required	true, for JavaScript mutators

type	String

YML

example	

```
eval: 'return JSON.stringify({"some stuff": "is here"});'
```

JSON

```
{
  "eval": "return JSON.stringify({\"some info\": \"is here\"});"
}
```

runtime_assets

description	Array of Sensu dynamic runtime assets (by their names) required at runtime for execution of the `command`.
required	false
type	Array

YML

example	

```
runtime_assets:
- metric-mutator
```

JSON

```
{
  "runtime_assets": [
    "metric-mutator"
  ]
}
```

secrets

description	Array of the name/secret pairs to use with command execution.
required	false
type	Array

example

YML

```
secrets:
- name: ANSIBLE_HOST
  secret: sensu-ansible-host
- name: ANSIBLE_TOKEN
  secret: sensu-ansible-token
```

JSON

```
{
  "secrets": [
    {
      "name": "ANSIBLE_HOST",
      "secret": "sensu-ansible-host"
    },
    {
      "name": "ANSIBLE_TOKEN",
      "secret": "sensu-ansible-token"
    }
  ]
}
```

timeout

description

Mutator execution duration timeout (hard stop). In seconds.

***WARNING**: The timeout attribute is available for JavaScript mutators but may not work properly if the mutator is in a loop.*

required	false

type	integer

YML

example	`timeout: 30`

JSON

```
{
  "timeout": 30
}
```

type

description	Mutator type.

> ***NOTE**: Make sure to specify the type is `javascript` when you create a JavaScript mutator. If you do not specify the type, Sensu uses `pipe` as the default, expects a command attribute in the mutator definition, and ignores any eval attribute you provide.*

required	false
type	String
default	`pipe`
allowed values	`pipe` and `javascript`

YML

example	`type: pipe`

JSON

```
{
  "type": "pipe"
}
```

```
}
```

`secrets` *attributes*

name	
description	Name of the secret defined in the executable command. Becomes the environment variable presented to the mutator. Read Use secrets management in Sensu for more information.
required	true
type	String
example	**YML** `name: ANSIBLE_HOST` **JSON** `{ "name": "ANSIBLE_HOST" }`

secret	
description	Name of the Sensu secret resource that defines how to retrieve the secret.
required	true
type	String
example	**YML** `secret: sensu-ansible-host`

JSON

```
{
  "secret": "sensu-ansible-host"
}
```

Use secrets management in a mutator

Learn more about secrets management for your Sensu configuration in the secrets and secrets providers references.

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: ansible-tower
  namespace: ops
spec:
  command: sensu-ansible-mutator -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN
  secrets:
  - name: ANSIBLE_HOST
    secret: sensu-ansible-host
  - name: ANSIBLE_TOKEN
    secret: sensu-ansible-token
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "ansible-tower",
    "namespace": "ops"
  },
```

```
  "spec": {
    "command": "sensu-ansible-mutator -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN",
    "secrets": [
      {
        "name": "ANSIBLE_HOST",
        "secret": "sensu-ansible-host"
      },
      {
        "name": "ANSIBLE_TOKEN",
        "secret": "sensu-ansible-token"
      }
    ]
  }
}
```

Add new event attributes with JavaScript mutators

Use a JavaScript mutator to rewrite events with a new attribute added.

This example adds a new “organization” attribute to events at the top level, with a value of `sec_ops`:

YML

```
---
type: Mutator
api_version: core/v2
metadata:
  name: add_org_sec_ops
spec:
  eval: >-
    data = JSON.parse(JSON.stringify(event)); data['organization'] = 'sec_ops';
    return JSON.stringify(data)
  type: javascript
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
```

```
  "metadata": {
    "created_by": "admin",
    "name": "add_org_sec_ops"
  },
  "spec": {
    "eval": "data = JSON.parse(JSON.stringify(event)); data['organization'] =
'sec_ops'; return JSON.stringify(data)",
    "type": "javascript"
  }
}
```

Combine existing attributes with JavaScript mutators

Use a JavaScript mutator to create a new attribute from a combination of multiple existing attributes and add the new attribute to events.

This example combines the event namespace and the name of the check that generated the event into a single new attribute, `origination`:

YML

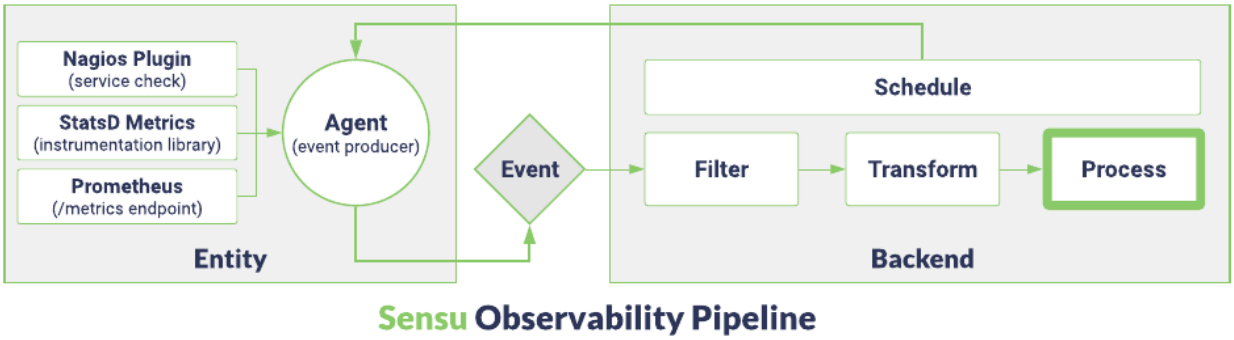
```
---
type: Mutator
api_version: core/v2
metadata:
  name: add_origination_attribute
spec:
  eval: >-
    data = JSON.parse(JSON.stringify(event)); data.origination =
    data.metadata.namespace + data.check.metadata.name; return
    JSON.stringify(data)
  type: javascript
```

JSON

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "add_origination_attribute"
  },
  "spec": {
    "eval": "data = JSON.parse(JSON.stringify(event)); data.origination =
data.metadata.namespace + data.check.metadata.name; return JSON.stringify(data)",
    "type": "javascript"
  }
}
```

Process your observation data



Sensu Observability Pipeline

Back to start **or click any element in the pipeline to jump to it.**

In the process stage, Sensu executes pipelines and handlers.

In the process stage of Sensu's observability pipeline, the Sensu backend executes pipelines and handlers to take action on your observation data. Your pipeline or handler configuration determines what happens to the events that comes through your observability pipeline. For example, your pipeline or handler might route incidents to a specific Slack channel or PagerDuty notification workflow, or send metrics to InfluxDB or Prometheus.

Pipelines

Pipelines are Sensu resources composed of observation event processing workflows made up of filters, mutators, and handlers. Instead of specifying filters and mutators in handler definitions, you can specify all three in a single pipeline workflow.

This example shows a pipeline resource definition that includes an event filter, a mutator, and a handler:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: incident_alerts
```

```
spec:
  workflows:
  - name: labeled_email_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    mutator:
      name: add_labels
      type: Mutator
      api_version: core/v2
    handler:
      name: email
      type: Handler
      api_version: core/v2
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "labeled_email_alerts",
        "filters": [
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
```

```
            "name": "email",
            "type": "Handler",
            "api_version": "core/v2"
          }
        }
      ]
    }
}
```

To use pipelines, list them in check definitions in the pipelines array. All the observability events that the check produces will be processed according to the pipeline's workflows.

Handlers

Handlers are actions the Sensu backend executes on events. Sensu also checks your handlers for the event filters and mutators to apply in the filter and transform stages.

A few different types of handlers are available in Sensu. The most common are pipe handlers, which work similarly to checks and enable Sensu to interact with almost any computer program via standard streams.

Here's an example resource definition for a pipe handler — read Send Slack alerts with handlers to learn how to configure your own version of this handler:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: slack
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  runtime_assets:
  - sensu-slack-handler
  secrets: null
  timeout: 0
```

```
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX"
    ],
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

Other types of handlers include Sumo Logic metrics handlers and TCP stream handlers, which provide persistent connections for transmitting Sensu observation data to remote data storage services to help prevent data bottlenecks.Sensu's Sumo Logic metrics handlers and TCP stream handlers are available for use **only** in pipelines.

You can also use traditional TCP/UDP handlers to send your observation data to remote sockets and handler sets to streamline groups of actions to execute for certain types of events.

Discover, download, and share Sensu handler dynamic runtime assets in Bonsai, the Sensu asset hub Read Use assets to install plugins to get started.

Handlers reference

Sensu executes handlers during the **process** stage of the observability pipeline.

Handlers are actions the Sensu backend executes on events.Several types of handlers are available. The most common are `pipe` handlers, which work similarly to checks and enable Sensu to interact with almost any computer program via standard streams.

- **Pipe handlers** send observation data (events) into arbitrary commands via stdin
- **TCP/UDP handlers** send observation data (events) to a remote socket
- **Handler sets** group event handlers and streamline groups of actions to execute for certain types of events (also called “set handlers”)

The handler stack concept describes a group of handlers or a handler set that escalates events through a series of different handlers.

Discover, download, and share Sensu handler dynamic runtime assets using Bonsai, the Sensu asset hub.Read Use dynamic runtime assets to install plugins to get started.

Pipe handlers

Pipe handlers are external commands that can consume event data via stdin.

Pipe handler example

This example shows a pipe handler resource definition with the minimum required attributes:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: pipe_handler_minimum
spec:
```

```
  command: command-example
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "pipe_handler_minimum"
  },
  "spec": {
    "command": "command-example",
    "type": "pipe"
  }
}
```

Pipe handler command

Pipe handler definitions include a `command` attribute, which is a command for the Sensu backend to execute.

Pipe handler command arguments

Pipe handler `command` attributes may include command line arguments for controlling the behavior of the `command` executable.

TCP/UDP handlers

TCP and UDP handlers enable Sensu to forward event data to arbitrary TCP or UDP sockets for external services to consume.

TCP/UDP handler example

This handler will send event data to a TCP socket (10.0.1.99:4444) and timeout if an acknowledgement (`ACK`) is not received within 30 seconds:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: tcp_handler
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: tcp
  timeout: 30
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "tcp_handler"
  },
  "spec": {
    "type": "tcp",
    "timeout": 30,
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Change the `type` from `tcp` to `udp` to configure a UDP handler:

YML

```
---
type: Handler
api_version: core/v2
metadata:
```

```
  name: udp_handler
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: udp
  timeout: 30
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "udp_handler"
  },
  "spec": {
    "type": "udp",
    "timeout": 30,
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Handler sets

> ***NOTE**: We recommend using pipelines to configure multiple workflows for different handlers instead of handler sets.*

Handler set definitions allow you to use a single named handler set to refer to groups of handlers. The handler set becomes a collection of individual actions to take (via each included handler) on event data.

For example, suppose you have already created these two handlers:

- `elasticsearch` to send all observation data to Elasticsearch.

- `opsgenie` to send non-OK status alerts to your OpsGenie notification channel.

You can list both of these handlers in a handler set to automate and streamline your workflow, specifying `type: set`:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: send_events_notify_operator
spec:
  handlers:
  - elasticsearch
  - opsgenie
  type: set
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "send_events_notify_operator"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "elasticsearch",
      "opsgenie"
    ]
  }
}
```

Now you can route observation data to Elasticsearch and alerts to OpsGenie with a single handler definition, the `send_events_notify_operator` handler set.

***NOTE**: Attributes defined in handler sets do not apply to the handlers they include. For example,*

`filters` and `mutator` attributes defined in a handler set will have no effect on handlers. Define these attributes in individual handlers instead, or use pipelines.

Handler stacks

***NOTE**: We recommend using pipelines to configure multiple workflows for escalating events through a series of handlers instead of handler stacks.*

The handler stack concept refers to a group of handlers or a handler set that escalates events through a series of different handlers. For example, suppose you want a handler stack with three levels of escalation:

- Level 1: On the first occurrence, attempt remediation.
- Level 2: On the fifth occurrence, send an alert to Slack.
- Level 3: On the tenth occurrence, send an alert to PagerDuty. Continue to send this alert on every tenth occurrence thereafter until the incident is resolved.

A handler stack for this scenario requires three handlers to take the desired actions based on three corresponding event filters that control the escalation levels:

- Level 1 requires an event filter with the built-in `is_incident` filter plus an occurrence-based filter that uses an expression like `event.check.occurrences ==1` and a corresponding remediation handler.
- Level 2 requires an event filter with `is_incident` plus an occurrence-based filter that uses an expression like `event.check.occurrences == 5` and a corresponding Slack handler.
- Level 3 requires an event filter with `is_incident` plus an occurrence-based filter that uses an expression like `event.check.occurrences % 10 == 0` to match event data with an occurrences value that is evenly divisible by 10 via a modulo operator calculation and a corresponding PagerDuty handler.

With these event filters and handlers configured, you can create a handler set that includes the three handlers in your stack. You can also list the three handlers in the handlers array in your check definition instead.

***PRO TIP**: This scenario relies on six different resources, three event filters and three handlers, to describe the handler stack concept, but you can use Sensu dynamic runtime assets and integrations to achieve the same escalating alert levels in other ways.*

For example, you can use the `is_incident` event filter in conjunction with the sensu/sensu-go-fatigue-check-filter asset to control event escalation. The sensu/sensu-ansible-handler, sensu/sensu-rundeck-handler, and sensu/sensu-saltstack-handler auto-remediation integrations and the sensu/sensu-remediation-handler asset also include built-in occurrence- and severity-based event filtering.

Keepalive event handlers

Sensu keepalives are the heartbeat mechanism used to ensure that all registered Sensu agents are operational and can reach the Sensu backend. You can connect keepalive events to your monitoring workflows using a keepalive handler. Sensu looks for an event handler named `keepalive` and automatically uses it to process keepalive events.

Suppose you want to receive Slack notifications for keepalive alerts, and you already have a Slack handler set up to process events. To process keepalive events using the Slack pipeline, create a handler set named `keepalive` and add the `slack` handler to the `handlers` array. The resulting `keepalive` handler set configuration will look like this example:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: keepalive
spec:
  handlers:
  - slack
  type: set
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "keepalive"
  },
  "spec": {
```

```
    "type": "set",
    "handlers": [
      "slack"
    ]
  }
}
```

You can also use the `keepalive-handlers` configuration option to send keepalive events to any handler you have configured.If you do not specify a keepalive handler with the `keepalive-handlers` configuration option, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI.

Handler specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For handlers in this version of Sensu, the `api_version` should always be `core/v2`.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata	
description	Top-level collection of metadata about the handler that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the handler definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: handler-slack` `  namespace: default` `  created_by: admin` `  labels:` `  region: us-west-1` `  annotations:` `  slack-channel: "#monitoring"` **JSON** `{` `  "metadata": {` `  "name": "handler-slack",` `  "namespace": "default",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1"` `  },` `  "annotations": {` `  "slack-channel": "#monitoring"` `  }` `  }` `}`

spec	
description	Top-level map that includes the handler spec attributes.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** ```yaml spec: type: tcp socket: host: 10.0.1.99 port: 4444 metadata: name: tcp_handler namespace: default ``` **JSON** ```json { "spec": { "type": "tcp", "socket": { "host": "10.0.1.99", "port": 4444 }, "metadata": { "name": "tcp_handler", "namespace": "default" } } } ```

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Handlers should always be type `Handler`.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** ```yaml type: Handler ``` **JSON** ```json { "type": "Handler" } ```

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** ```yaml annotations: ```

```
  managed-by: ops
  playbook: www.example.url
```

JSON

```
{
  "annotations": {
    "managed-by": "ops",
    "playbook": "www.example.url"
  }
}
```

created_by

description	Username of the Sensu user who created the handler or last updated the handler. Sensu automatically populates the `created_by` field when the handler is created or updated.
required	false
type	String

YML

example

```
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation event data that you can

	use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

name

description	Unique string used to identify the handler. Handler names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each handler must have a unique name within its namespace.
required	true

type	String
example	**YML** `name: handler-slack` **JSON** `{ "name": "handler-slack" }`

namespace

description	Sensu RBAC namespace that the handler belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: production` **JSON** `{ "namespace": "production" }`

Spec attributes

command

description	Handler command to be executed. The event data is passed to the process via stdin. ***NOTE**: The `command` attribute is only supported for pipe handlers (that is, handlers configured with `"type": "pipe"`).*
required	true (if `type` equals `pipe`)
type	String
example	**YML** `command: /etc/sensu/plugins/pagerduty.go` **JSON** `{ "command": "/etc/sensu/plugins/pagerduty.go" }`

env_vars

description	Array of environment variables to use with command execution. ***NOTE**: The `env_vars` attribute is only supported for pipe handlers (that is, handlers configured with `"type": "pipe"`).*
required	false
type	Array
example	**YML** `env_vars:` `- API_KEY=0428d6b8nb51an4d95nbe28nf90865a66af5`

JSON

```
{
  "env_vars": [
    "API_KEY=0428d6b8nb51an4d95nbe28nf90865a66af5"
  ]
}
```

filters

description	Array of Sensu event filters (by names) to use when filtering events for the handler. Each array item must be a string. ***NOTE**: We recommend using pipelines, which allow you to list event filters directly in the pipeline resource definition instead of in handlers.* *Pipelines ignore any event filters specified in handler definitions, so you do not need to remove them to use your existing handlers — just make sure to define the event filters you want to use in the pipeline workflow.*
required	false
type	Array
example	**YML** `filters:` `- is_incident` `- not_silenced` `- state_change_only` **JSON** `{` `"filters": [`

```
    "is_incident",
    "not_silenced",
    "state_change_only"
  ]
}
```

handlers

description	Array of Sensu event handlers (by their names) to use for events using the handler set. Each array item must be a string. ***NOTE**: The `handlers` attribute is only supported for handler sets (that is, handlers configured with `"type": "set"`).* *We recommend using pipelines to configure multiple workflows instead of handler sets.*
required	true (if `type` equals `set`)
type	Array
example	**YML** `handlers:` `- pagerduty` `- email` `- ec2` **JSON** `{ "handlers": [ "pagerduty", "email", "ec2" ] }`

mutator

description	Name of the Sensu event mutator to use to mutate event data for the handler. ***NOTE**: We recommend using pipelines, which allow you to list mutators directly in the pipeline resource definition instead of in handlers.* *Pipelines ignore any mutators specified in handler definitions, so you do not need to remove them to use your existing handlers — just make sure to define the mutator you want to use in the pipeline workflow.*
required	false
type	String
example	**YML** `mutator: only_check_output` **JSON** `{ "mutator": "only_check_output" }`

runtime_assets

description	Array of Sensu dynamic runtime assets (by names) required at runtime to execute the `command`
required	false

type	Array

YML

example

```
runtime_assets:
- metric-handler
```

JSON

```
{
  "runtime_assets": [
    "metric-handler"
  ]
}
```

secrets

description	Array of the name/secret pairs to use with command execution.
required	false
type	Array

YML

example

```
secrets:
- name: ANSIBLE_HOST
  secret: sensu-ansible-host
- name: ANSIBLE_TOKEN
  secret: sensu-ansible-token
```

JSON

```
{
  "secrets": [
    {
      "name": "ANSIBLE_HOST",
      "secret": "sensu-ansible-host"
    },
```

```
    {
      "name": "ANSIBLE_TOKEN",
      "secret": "sensu-ansible-token"
    }
  ]
}
```

socket

description

Scope for `socket` definition used to configure the TCP/UDP handler socket.

> ***NOTE**: The `socket` attribute is only supported for TCP/UDP handlers (that is, handlers configured with `"type": "tcp"` or `"type": "udp"`).*

required

true (if `type` equals `tcp` or `udp`)

type

Hash

example

YML

```
socket: {}
```

JSON

```
{
  "socket": {}
}
```

timeout

description

Handler execution duration timeout (hard stop). In seconds. Only used by `pipe`, `tcp`, and `udp` handler types.

required	false
type	Integer
default	`60` (for `tcp` and `udp` handlers)
example	**YML** ```yaml timeout: 30 ``` **JSON** ```json { "timeout": 30 } ```

type

description	Handler type.
required	true
type	String
allowed values	`pipe`, `tcp`, `udp`, and `set`
example	**YML** ```yaml type: pipe ``` **JSON** ```json { "type": "pipe" } ```

`socket` *attributes*

host

description	Socket host address (IP or hostname) to connect to.
required	true
type	String
example	**YML** `host: 8.8.8.8` **JSON** `{ "host": "8.8.8.8" }`

port

description	Socket port to connect to.
required	true
type	Integer
example	**YML** `port: 4242` **JSON** `{ "port": 4242 }`

`secrets` *attributes*

name	
description	Name of the secret defined in the executable command. Becomes the environment variable presented to the handler. Read Use secrets management in Sensu for more information.
required	true
type	String
example	**YML** `name: ANSIBLE_HOST` **JSON** `{ "name": "ANSIBLE_HOST" }`

secret	
description	Name of the Sensu secret resource that defines how to retrieve the secret.
required	true
type	String
example	**YML** `secret: sensu-ansible-host` **JSON**

```
{
  "secret": "sensu-ansible-host"
}
```

Send Slack alerts

This handler will send alerts to a channel named `monitoring` with the configured webhook URL, using the `handler-slack` executable command. The handler uses the sensu/sensu-slack-handler dynamic runtime asset. Read Send Slack alerts with handlers for detailed instructions for adding the required asset and configuring this handler.

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: slack
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXX
XXXXXXXXXX
  handlers: []
  runtime_assets:
  - sensu/sensu-slack-handler
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack"
```

```
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
    ],
    "handlers": [],
    "runtime_assets": [
      "sensu/sensu-slack-handler"
    ],
    "timeout": 0,
    "type": "pipe"
  }
}
```

Send registration events

If you configure a Sensu event handler named `registration`, the Sensu backend will create and process an event for the agent registration, apply any configured filters and mutators, and execute the registration handler.

Read Automatically register and deregister entities for more information and a registration handler example.

Execute multiple handlers (handler set)

> ***NOTE**: We recommend using pipelines to configure multiple workflows for different handlers instead of handler sets.*

The following example creates a handler set, `notify_all_the_things`, that will execute three handlers: `slack`, `tcp_handler`, and `udp_handler`.

YML

```
---
```

```
type: Handler
api_version: core/v2
metadata:
  name: notify_all_the_things
spec:
  handlers:
  - slack
  - tcp_handler
  - udp_handler
  type: set
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "notify_all_the_things"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack",
      "tcp_handler",
      "udp_handler"
    ]
  }
}
```

Use secrets management in a handler

Learn more about secrets management for your Sensu configuration in the secrets and secrets providers references.

YML

```
---
type: Handler
api_version: core/v2
```

```
metadata:
  name: ansible-tower
spec:
  type: pipe
  command: sensu-ansible-handler -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN
  secrets:
  - name: ANSIBLE_HOST
    secret: sensu-ansible-host
  - name: ANSIBLE_TOKEN
    secret: sensu-ansible-token
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "ansible-tower"
  },
  "spec": {
    "type": "pipe",
    "command": "sensu-ansible-handler -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN",
    "secrets": [
      {
        "name": "ANSIBLE_HOST",
        "secret": "sensu-ansible-host"
      },
      {
        "name": "ANSIBLE_TOKEN",
        "secret": "sensu-ansible-token"
      }
    ]
  }
}
```

Pipelines reference

***IMPORTANT**: The pipelines described on this page are different from the resources you can create and manage with the `enterprise/pipeline/v1` API. The `enterprise/pipeline/v1` API allows you to create and manage resources that can **only** be used in pipelines rather than pipelines themselves.*

Read the Sumo Logic metrics handlers reference and TCP stream handlers reference for more information about enterprise pipeline resources.

Sensu executes pipelines during the **process** stage of the observability pipeline.

Pipelines are Sensu resources composed of observation event processing workflows made up of filters, mutators, and handlers. Instead of specifying filters and mutators in handler definitions, you can specify all three in a single pipeline workflow.

To use a pipeline, list it in a check definition's pipelines array. All the observability events that the check produces will be processed according to the pipeline's workflows.

Pipelines can replace handler sets and handler stacks. We recommend migrating your existing handler sets and stacks to pipeline workflows.

***NOTE**: To use pipelines, upgrade your agents to Sensu Go 6.5.0.*

Pipeline example

This example shows a pipeline resource definition that includes event filters, a mutator, and a handler:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: incident_alerts
```

```
spec:
  workflows:
  - name: labeled_email_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    - name: state_change_only
      type: EventFilter
      api_version: core/v2
    mutator:
      name: add_labels
      type: Mutator
      api_version: core/v2
    handler:
      name: email
      type: Handler
      api_version: core/v2
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "labeled_email_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
```

```
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

To use this pipeline in a check, list it in the check's pipelines array.For example:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: incident_pipelines
spec:
  command: collect.sh
  interval: 10
  publish: true
  subscriptions:
  - system
  pipelines:
  - type: Pipeline
```

```
    api_version: core/v2
    name: incident_alerts
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_pipelines"
  },
  "spec": {
    "command": "collect.sh",
    "interval": 10,
    "publish": true,
    "subscriptions": [
      "system"
    ],
    "pipelines": [
      {
        "type": "Pipeline",
        "api_version": "core/v2",
        "name": "incident_alerts"
      }
    ]
  }
}
```

Workflows

The workflow attribute is an array of event processing workflows that Sensu will apply for events produced by any check that references the pipeline.

Workflows do not have to include an event filter or mutator, but they must specify at least one handler.

Workflows can include more than one event filter. If a workflow has more than one filter, Sensu applies the filters in a series, starting with the filter that is listed first.

You can use your existing event filters, mutators, and handlers in pipeline workflows. Pipelines ignore

any filters and mutators specified in handler definitions, so you do not need to remove them to use your existing handlers — just make sure to define the event filters and mutators you want to use in the pipeline workflow.

Pipelines with multiple workflows

Pipelines can include more than one workflow.

In this example, the pipeline includes `labeled_email_alerts` and `slack_alerts` workflows:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: incident_alerts
spec:
  workflows:
  - name: labeled_email_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    - name: state_change_only
      type: EventFilter
      api_version: core/v2
    mutator:
      name: add_labels
      type: Mutator
      api_version: core/v2
    handler:
      name: email
      type: Handler
      api_version: core/v2
  - name: slack_alerts
    filters:
    - name: is_incident
```

```
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    - name: state_change_only
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "labeled_email_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
```

```
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      },
      {
        "name": "slack_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

All events from checks that specify this pipeline will be processed with both workflows, in series, starting with the workflow that is listed first in the resource definition.

Read Route alerts with event filters for another pipeline example that includes multiple workflows for contact-based routing.

Pipeline specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For pipelines in this version of Sensu, the api_version should always be `core/v2`.
required	Required for pipeline definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata	
description	Top-level collection of metadata about the pipeline that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the

	pipeline definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for pipeline definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: incident_alerts` `  namespace: default` `  created_by: admin` `  labels:` `  region: us-west-1` `  annotations:` `  slack-channel: "#incidents"` **JSON** `{` `  "metadata": {` `  "name": "incident_alerts",` `  "namespace": "default",` `  "created_by": "admin",` `  "labels": {` `  "region": "us-west-1"` `  },` `  "annotations": {` `  "slack-channel": "#incidents"` `  }` `  }` `}`

spec

description	Top-level map that includes the pipeline spec attributes.

required	Required for pipeline definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** ```yaml spec: workflows: - name: labeled_email_alerts filters: - name: is_incident type: EventFilter api_version: core/v2 - name: not_silenced type: EventFilter api_version: core/v2 - name: state_change_only type: EventFilter api_version: core/v2 mutator: name: add_labels type: Mutator api_version: core/v2 handler: name: email type: Handler api_version: core/v2 ``` **JSON** ```json { "spec": { "workflows": [{ "name": "labeled_email_alerts", "filters": [{ "name": "is_incident", "type": "EventFilter", "api_version": "core/v2" }, { ```

```
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Pipelines should always be type `Pipeline`.
required	Required for pipeline definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	YML `type: Pipeline`

JSON

```
{
  "type": "Pipeline"
}
```

Metadata attributes

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` `  slack-channel: "#incidents"` **JSON** `{` `  "annotations": {` `  "managed-by": "ops",` `  "slack-channel": "#incidents"` `  }` `}`

created_by

description	Username of the Sensu user who created the pipeline or last updated the handler. Sensu automatically populates the `created_by` field when the pipeline is created or updated.
required	false
type	String

YML

example

```
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8

string.

default `null`

example

YML

```yaml
labels:
  environment: production
  region: us-west-1
```

JSON

```json
{
  "labels": {
    "environment": "production",
    "region": "us-west-1"
  }
}
```

name

description

Unique string used to identify the pipeline. Pipeline names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each pipeline must have a unique name within its namespace.

required

true

type

String

example

YML

```yaml
name: incident_alerts
```

JSON

```json
{
  "name": "incident_alerts"
}
```

namespace	
description	Sensu RBAC namespace that the pipeline belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: default` **JSON** `{ "namespace": "default" }`

Spec attributes

workflows	
description	Array of workflows (by names) to use when filtering, mutating, and handling observability events with a pipeline. Each array item must be a string. Read workflows attributes for details.
required	false
type	Array
example	**YML** `workflows:` `  - name: labeled_email_alerts` `  filters:`

```
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    - name: state_change_only
      type: EventFilter
      api_version: core/v2
  mutator:
    name: add_labels
    type: Mutator
    api_version: core/v2
  handler:
    name: email
    type: Handler
    api_version: core/v2
```

JSON

```
{
  "workflows": [
    {
      "name": "labeled_email_alerts",
      "filters": [
        {
          "name": "is_incident",
          "type": "EventFilter",
          "api_version": "core/v2"
        },
        {
          "name": "not_silenced",
          "type": "EventFilter",
          "api_version": "core/v2"
        },
        {
          "name": "state_change_only",
          "type": "EventFilter",
          "api_version": "core/v2"
        }
      ],
```

```
      "mutator": {
        "name": "add_labels",
        "type": "Mutator",
        "api_version": "core/v2"
      },
      "handler": {
        "name": "email",
        "type": "Handler",
        "api_version": "core/v2"
      }
    }
  ]
}
```

Workflows attributes

filters	
description	Reference for the Sensu event filters to use when filtering events for the pipeline. Each pipeline workflow can reference more than one event filter. If a workflow has more than one filter, Sensu applies the filters in a series, starting with the filter that is listed first. Read filters attributes for details.
required	false
type	Map of key-value pairs
default	`null`
example	YML `filters:` `- name: is_incident` `  type: EventFilter` `  api_version: core/v2` `- name: not_silenced` `  type: EventFilter` `  api_version: core/v2`

```
- name: state_change_only
  type: EventFilter
  api_version: core/v2
```

JSON

```
{
  "filters": [
    {
      "name": "is_incident",
      "type": "EventFilter",
      "api_version": "core/v2"
    },
    {
      "name": "not_silenced",
      "type": "EventFilter",
      "api_version": "core/v2"
    },
    {
      "name": "state_change_only",
      "type": "EventFilter",
      "api_version": "core/v2"
    }
  ]
}
```

handler

description	Reference for the Sensu handler to use for event processing in the workflow. Each pipeline workflow must reference one handler. Pipelines ignore any filters and mutators specified in handler definitions. Read handler attributes for details.
required	true
type	Map of key-value pairs
example	**YML**

```
handler:
  name: email
  type: Handler
  api_version: core/v2
```

JSON

```
{
  "handler": {
    "name": "email",
    "type": "Handler",
    "api_version": "core/v2"
  }
}
```

mutator	
description	Reference for the Sensu mutator to use to mutate event data for the workflow. Each pipeline workflow can reference only one mutator. Read mutator attributes for details.
required	false
type	Map of key-value pairs
default	`null`
example	**YML** `mutator:` `  name: add_labels` `  type: Mutator` `  api_version: core/v2` **JSON** `{` `  "mutator": {` `  "name": "add_labels",`

```
    "type": "Mutator",
    "api_version": "core/v2"
  }
}
```

Filters attributes

api_version	
description	The Sensu API group and version for the event filter. For event filters in this version of Sensu, the api_version should always be `core/v2`.
required	true
type	String
default	`null`
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

name	
description	Name of the Sensu event filter to use for the workflow. You can use the built-in event filters, as well as your existing event filters, in pipeline workflows.
required	true

type	String
default	`null`
example	**YML** `name: is_incident` **JSON** `{ "name": "is_incident" }`

type

description	The `sensuctl create` resource type for the event filter. Event filters should always be type `EventFilter`.
required	true
type	String
default	`null`
example	**YML** `type: EventFilter` **JSON** `{ "type": "EventFilter" }`

Handler attributes

api_version

description	The Sensu API group and version for the handler.
required	true
type	String
allowed values	`core/v2` for a pipe handler, TCP or UDP handler, or handler set `pipeline/v1` for a TCP stream handler or Sumo Logic metrics handler
default	`null`
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

name

description	Name of the Sensu handler to use for the workflow. You can use your existing handlers in pipeline workflows — pipelines ignore any filters and mutators specified in handler definitions.
required	true
type	String
default	`null`
example	**YML** `name: email`

JSON

```
{
  "name": "email"
}
```

type	
description	The `sensuctl create` resource type for the handler.
required	true
type	String
allowed values	`Handler` for a pipe handler, TCP or UDP handler, or handler set `TCPStreamHandler` for a TCP stream handler `SumoLogicMetricsHandler` for a Sumo Logic metrics handler
default	`null`
example	**YML** `type: Handler` **JSON** `{ "type": "Handler" }`

Mutator attributes

api_version

description	The Sensu API group and version for the mutator. For mutators in this version of Sensu, the api_version should always be `core/v2`.
required	true
type	String
default	`null`
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

name

description	Name of the Sensu mutator to use for the workflow. You can use your existing mutators in pipeline workflows.
required	true
type	String
default	`null`
example	**YML** `name: add_labels` **JSON** `{ "name": "add_labels" }`

type	
description	The `sensuctl create` resource type for the mutator. Mutators should always be type `Mutator`.
required	true
type	String
default	`null`
example	**YML** `type: Mutator` **JSON** `{ "type": "Mutator" }`

Silencing reference

Sensu's silencing capability allows you to suppress event handler execution on an ad hoc basis so you can plan maintenance and reduce alert fatigue. Silences are created on an ad hoc basis using sensuctl, the web UI, and the core/v2/silenced API endpoints.

Successfully created silencing entries are assigned a `name` in the format `$SUBSCRIPTION:$CHECK`, where `$SUBSCRIPTION` is the name of a Sensu entity subscription and `$CHECK` is the name of a Sensu check. You can use silences to silence checks on specific entities by taking advantage of per-entity subscriptions (for example, `entity:$ENTITY_NAME`).

When creating a silencing entry, you can specify a combination of checks and subscriptions, but only one or the other is strictly required. For example, if you create a silencing entry specifying only a check, its name will contain an asterisk (or wildcard) in the `$SUBSCRIPTION` position. This indicates that any event with a matching check name will be marked as silenced, regardless of the originating entities' subscriptions.

Conversely, a silencing entry that specifies only a subscription will have a name with an asterisk in the `$CHECK` position. This indicates that any event where the originating entities' subscriptions match the subscription specified in the entry will be marked as silenced, regardless of the check name.

These silences are persisted in the Sensu datastore. When the Sensu server processes subsequent check results, it retrieves matching silences from the store. If there are one or more matching entries, the event is updated with a list of silenced entry names. When the check name or subscription described in a silencing entry matches an event, the event will include the `silenced` attribute, which lists the silencing entries that match the event.

Silenced checks still create events, and events from silenced checks are still passed to handlers. To prevent handler execution for events from silenced checks, make sure the handler definition includes the built-in `not_silenced` event filter. The `not_silenced` event filter prevents handlers from processing events that include the `silenced` attribute.

Silencing examples

This example shows a silencing resource definition that uses a per-entity subscription to silence any alerts on a single Sensu entity, `i-424242`:

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  name: entity:i-424242:*
spec:
  begin: 1542671205
  check: null
  creator: admin
  expire: -1
  expire_at: 0
  expire_on_resolve: false
  reason: null
  subscription: entity:i-424242
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "entity:i-424242:*"
  },
  "spec": {
    "expire": -1,
    "expire_at": 0,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": null,
    "check": null,
    "subscription": "entity:i-424242",
    "begin": 1542671205
  }
}
```

Silence a specific check on a specific entity

The following example shows how to silence a check named `check_ntp` on entity `i-424242`,

ensuring the silencing entry is deleted after the underlying issue is resolved:

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  name: entity:i-424242:check_ntp
spec:
  subscription: entity:i-424242
  check: check_ntp
  expire_on_resolve: true
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "entity:i-424242:check_ntp"
  },
  "spec": {
    "subscription": "entity:i-424242",
    "check": "check_ntp",
    "expire_on_resolve": true
  }
}
```

The optional `expire_on_resolve` attribute used in this example indicates that when the server processes a matching check from the specified entity with status OK, the silencing entry will be removed automatically.

When used in combination with other attributes (like `creator` and `reason`), this gives Sensu operators a way to acknowledge that they received an alert, suppress additional notifications, and automatically clear the silencing entry when the check status returns to normal.

Silencing specification

Silenced entry names

Silences must contain either a subscription or check name and are identified by the combination of `$SUBSCRIPTION:$CHECK`. If a check or subscription is not provided, it will be substituted with a wildcard (asterisk): `$SUBSCRIPTION:*` or `*:$CHECK`.

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Silences should always be type `Silenced`.
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: Silenced` **JSON** `{ "type": "Silenced" }`

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For silences in this version of Sensu, the `api_version` should always be `core/v2`.
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.

type	String

YML

example

```
api_version: core/v2
```

JSON

```
{
  "api_version": "core/v2"
}
```

metadata

description	Top-level collection of metadata about the silencing entry that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the silencing entry definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

example

```
metadata:
  name: appserver:mysql_status
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
```

JSON

```
{
  "metadata": {
    "name": "appserver:mysql_status",
```

```
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "region": "us-west-1"
    }
  }
}
```

spec	
description	Top-level map that includes the silencing entry spec attributes.
required	Required for silences in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  expire: -1` `  expire_at: 0` `  expire_on_resolve: false` `  creator: admin` `  reason:` `  check:` `  subscription: entity:i-424242` `  begin: 1542671205` **JSON** `{` `  "spec": {` `  "expire": -1,` `  "expire_at": 0,` `  "expire_on_resolve": false,` `  "creator": "admin",` `  "reason": null,` `  "check": null,`

```
    "subscription": "entity:i-424242",
    "begin": 1542671205
  }
}
```

Metadata attributes

name	
description	Silencing identifier generated from the combination of a subscription name and check name.
required	false - This value cannot be modified.
type	String
example	**YML** `name: appserver:mysql_status` **JSON** `{ "name": "appserver:mysql_status" }`

namespace	
description	Sensu RBAC namespace that the silencing entry belongs to.
required	false
type	String
default	`default`

example	**YML** ```yaml namespace: production ``` **JSON** ```json { "namespace": "production" } ```

created_by

description	Username of the Sensu user who created the silence or last updated the silence. Sensu automatically populates the `created_by` field when the silence is created or updated.
required	false
type	String
example	**YML** ```yaml created_by: admin ``` **JSON** ```json { "created_by": "admin" } ```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering.

	If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

annotations

description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.

required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** ```yaml annotations: managed-by: ops playbook: www.example.url ``` **JSON** ```json { "annotations": { "managed-by": "ops", "playbook": "www.example.url" } } ```

Spec attributes

check	
description	Name of the check the entry should match.
required	true, unless `subscription` is provided
type	String
example	**YML** ```yaml check: haproxy_status ``` **JSON** ```json { "check": "haproxy_status" ```

```
}
```

subscription

description	Name of the subscription the entry should match.
required	true, unless `check` is provided
type	String
example	**YML** `subscription: entity:i-424242` **JSON** `{ "subscription": "entity:i-424242" }`

begin

description	Time at which silence entry goes into effect. In epoch.
required	false
type	Integer
example	**YML** `begin: 1512512023` **JSON** `{ "begin": 1512512023 }`

expire

description	Number of seconds until the entry should be deleted. If the silence is set to expire when a check resolves, the `expire` value will be `-1`. If the silence is set to expire at a specific time, the `expire` value will be `0`.
required	false
type	Integer
default	-1
example	**YML** ```yaml expire: 3600 ``` **JSON** ```json { "expire": 3600 } ```

expire_at

description	Time at which the entry should be deleted. In seconds since the Unix epoch. Use `expire_at` in conjunction with `expire_on_resolve` to create silences that expire either when a check resolves or at a specific time, whichever comes first.

required	false
type	Integer
default	0
example	**YML** `expire_at: 1664550303` **JSON** `{ "expire_at": 1664550303 }`

expire_on_resolve

description	`true` if the entry should be deleted when the specified check begins to return OK status (resolves). Otherwise, `false`. Use `expire_on_resolve` in conjunction with `expire_at` to create silences that expire either when a check resolves or at a specific time, whichever comes first.
required	false
type	Boolean
default	false
example	**YML** `expire_on_resolve: true` **JSON** `{ "expire_on_resolve": true }`

creator

description	Person, application, or entity responsible for creating the entry.
required	false
type	String
default	null
example	**YML** `creator: Application Deploy Tool 5.0` **JSON** `{ "creator": "Application Deploy Tool 5.0" }`

reason

description	Explanation of the reason for creating the entry.
required	false
type	String
default	null
example	**YML** `reason: rebooting the world` **JSON**

```
{
  "reason": "rebooting the world"
}
```

Silence all checks with a specific subscription

Use this example to create a silencing entry for all checks with the `appserver` subscription:

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  name: appserver
spec:
  subscription: appserver
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "appserver"
  },
  "spec": {
    "subscription": "appserver"
  }
}
```

***NOTE**: This example will not silence entities with the `appserver` subscription.Checks that do not include the `appserver` subscription will still run on entities that include the `appserver` subscription.*

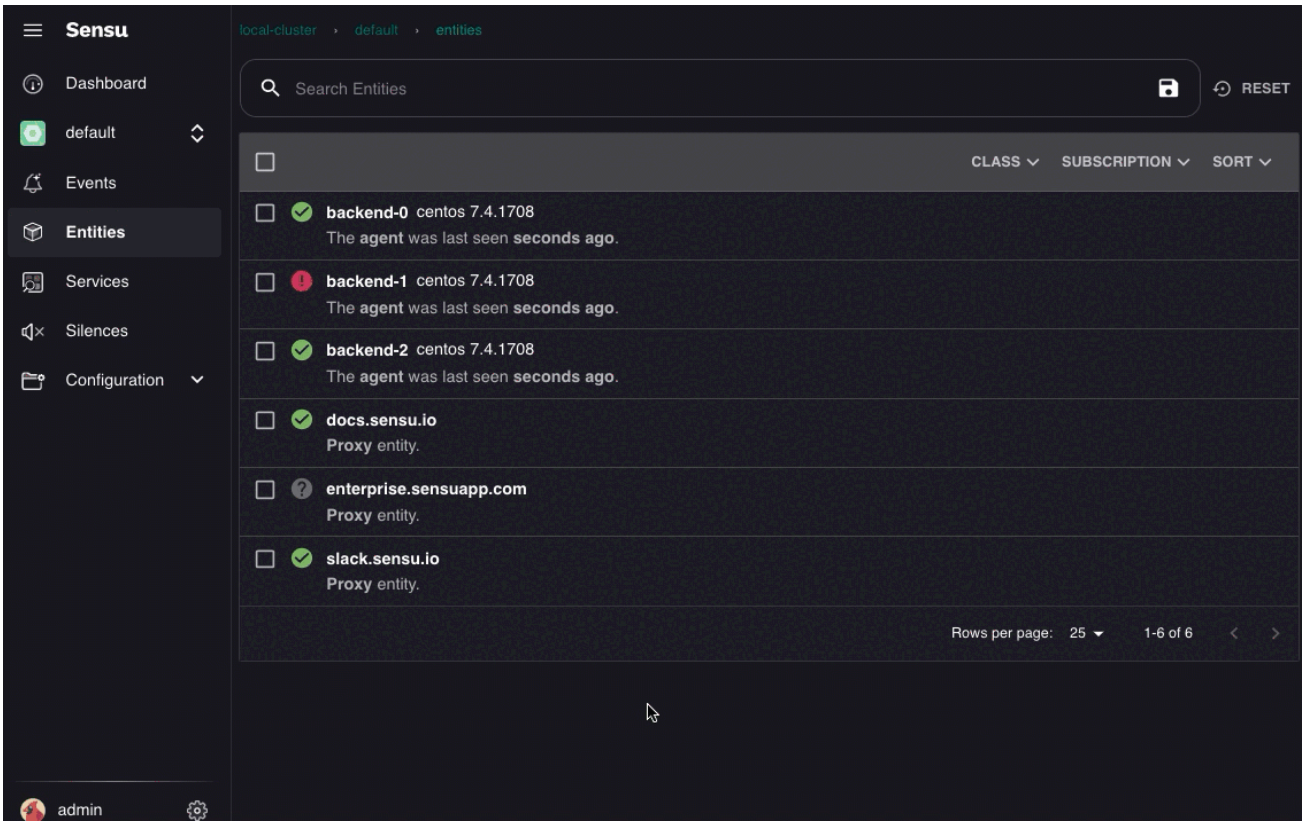
To silence all checks for entities with a particular subscription, use the Sensu web UI.

Silence all checks for entities with a specific subscription

To silence all checks for entities with a particular subscription:

1. Open the Entities page in the Sensu web UI.
2. Use the search field to search the entities by subscription. For example, to search for entities with the `system` subscription, enter `"system" in entity.subscriptions`.
3. Click the box to select all.
4. Click **SILENCE**.
5. In the New Silencing Entry dialog window, add any desired silence configuration options.
6. Click **CREATE**.

The silencing entries will be listed on the Silences page in the Sensu web UI.



Silence a specific check on entities with a specific subscription

To silence a check `mysql_status` that is running on Sensu entities with the subscription `appserver`:

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  name: appserver:mysql_status
spec:
  subscription: appserver
  check: mysql_status
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "appserver:mysql_status"
  },
  "spec": {
    "subscription": "appserver",
    "check": "mysql_status"
  }
}
```

Silence a specific check on every entity

To silence the check `mysql_status` on every entity in your infrastructure, regardless of subscriptions, you only need to provide the check name:

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  name: mysql_status
```

```
spec:
  check: mysql_status
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "mysql_status"
  },
  "spec": {
    "check": "mysql_status"
  }
}
```

Delete a silence

To delete a silencing entry, you must provide its name.

Subscription-only silencing entry names will contain an asterisk (or wildcard) in the `$SUBSCRIPTION` position, similar to this example:

YML

```
name: appserver:*
```

JSON

```
{
  "name": "appserver:*"
}
```

Check-only silencing entry names will contain an asterisk (or wildcard) in the `$CHECK` position, similar to this example:

YML

```
name: '*:mysql_status'
```

JSON

```
{
  "name": "*:mysql_status"
}
```

Sumo Logic metrics handlers reference

***COMMERCIAL FEATURE**: Access Sumo Logic metrics handlers in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu executes Sumo Logic metrics handlers during the **process** stage of the observability pipeline.

Sumo Logic metrics handlers provide a persistent connection to transmit Sensu observability metrics to a Sumo Logic HTTP Logs and Metrics Source, which helps prevent the data bottlenecks you may experience with traditional handlers.

Traditional handlers start a new UNIX process for every Sensu event they receive and require a new connection to send every event. As you scale up and process more events per second, the rate at which the handler can transmit observability event data decreases.

Sumo Logic metrics handlers allow you to configure a connection pool with a maximum number of connections for the handler to use and a time limit for request completion. For example, if 1000 events are queued for transmission, as each connection finishes transmitting an event, it becomes available again and returns to the pool so the handler can use it to send the next event in the queue.

Sumo Logic metrics handlers will reuse the available connections as long as they can rather than requiring a new connection for every event, which increases event throughput.

***NOTE**: Sumo Logic metrics handlers only accept metrics events. To send status events, use the Sensu Sumo Logic Handler integration instead.*

Sumo Logic metrics handler examples

This example shows a Sumo Logic metrics handler resource definition configured to send Sensu observability data to a Sumo Logic HTTP Logs and Metrics Source via the `url` attribute:

YML

```
---
type: SumoLogicMetricsHandler
```

```
api_version: pipeline/v1
metadata:
  name: sumologic_http_log_metrics
spec:
  url: "https://endpoint5.collection.us2.sumologic.com/receiver/v1/http/xxxxxxxx"
  max_connections: 10
  timeout: 30s
```

JSON

```
{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumologic_http_log_metrics"
  },
  "spec": {
    "url":
"https://endpoint5.collection.us2.sumologic.com/receiver/v1/http/xxxxxxxx",
    "max_connections": 10,
    "timeout": "30s"
  }
}
```

You can also use secrets management to avoid exposing the URL in your Sumo Logic metrics handler configuration:

YML

```
---
type: SumoLogicMetricsHandler
api_version: pipeline/v1
metadata:
  name: sumologic_http_log_metrics
spec:
  url: $SUMO_LOGIC_SOURCE_URL
  secrets:
  - name: SUMO_LOGIC_SOURCE_URL
    secret: sumologic_metrics_us2
  max_connections: 10
```

```
  timeout: 30s
```

JSON

```
{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumologic_http_log_metrics"
  },
  "spec": {
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
        "secret": "sumologic_metrics_us2"
      }
    ],
    "max_connections": 10,
    "timeout": "30s"
  }
}
```

Use Sumo Logic metrics handlers

Sumo Logic metrics handlers are commercial resources and are available for use **only** in pipelines.

> ***NOTE**: Sumo Logic metrics handlers **are not** used by listing the handler name in the check handlers attribute.*

To use a Sumo Logic metrics handler, list it as the handler in a pipeline definition. For example, this pipeline definition uses the sumologic_http_log_metrics example along with the built-in has_metrics event filter:

YML

```
---
type: Pipeline
```

```yaml
api_version: core/v2
metadata:
  name: metrics_workflows
spec:
  workflows:
  - name: metrics_to_sumologic
    filters:
    - name: has_metrics
      type: EventFilter
      api_version: core/v2
    handler:
      name: sumologic_http_log_metrics
      type: SumoLogicMetricsHandler
      api_version: pipeline/v1
```

JSON

```json
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "metrics_workflows"
  },
  "spec": {
    "workflows": [
      {
        "name": "metrics_to_sumologic",
        "filters": [
          {
            "name": "has_metrics",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "sumologic_http_log_metrics",
          "type": "SumoLogicMetricsHandler",
          "api_version": "pipeline/v1"
        }
      }
    ]
  }
```

```
}
```

Sumo Logic metrics handler specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Sumo Logic metrics handlers should always be type `SumoLogicMetricsHandler`.
required	Required for Sumo Logic metrics handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: SumoLogicMetricsHandler` **JSON** `{ "type": "SumoLogicMetricsHandler" }`

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For Sumo Logic metrics handlers in this version of Sensu, the `api_version` should always be `pipeline/v1`.
required	Required for Sumo Logic metrics handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.

type	String

YML

example	`api_version: pipeline/v1`

JSON

```
{
  "api_version": "pipeline/v1"
}
```

metadata

description	Top-level collection of metadata about the Sumo Logic metrics handler that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the handler definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for Sumo Logic metrics handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

example	`metadata:` `  name: sumologic_http_log_metrics` `  namespace: default` `  created_by: admin` `  labels:` `  environment: development` `  region: us-west-2` `  annotations:` `  managed-by: ops`

JSON

```
{
  "metadata": {
    "name": "sumologic_http_log_metrics",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "environment": "development",
      "region": "us-west-2"
    },
    "annotations": {
      "managed-by": "ops"
    }
  }
}
```

spec	
description	Top-level map that includes the Sumo Logic metrics handler spec attributes.
required	Required for Sumo Logic metrics handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  url: $SUMO_LOGIC_SOURCE_URL` `  secrets:` `  - name: SUMO_LOGIC_SOURCE_URL` `  secret: sumologic_metrics_us2` `  max_connections: 10` `  timeout: 30s` **JSON** `{` `  "spec": {`

```
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
        "secret": "sumologic_metrics_us2"
      }
    ],
    "max_connections": 10,
    "timeout": "30s"
  }
}
```

Metadata attributes

name	
description	Unique string used to identify the Sumo Logic metrics handler. Sumo Logic metrics handler names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each Sumo Logic metrics handler must have a unique name within its namespace.
required	true
type	String
example	**YML** `name: sumologic_http_log_metrics` **JSON** `{ "name": "sumologic_http_log_metrics" }`

namespace

description	Sensu RBAC namespace that the Sumo Logic metrics handler belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: default` **JSON** `{ "namespace": "default" }`

created_by

description	Username of the Sensu user who created the Sumo Logic metrics handler or last updated the Sumo Logic metrics handler. Sensu automatically populates the `created_by` field when the Sumo Logic metrics handler is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

```
}
```

labels	
description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` **JSON** `{` `  "annotations": {` `  "managed-by": "ops"` `  }` `}`

Spec attributes

url	
description	The URL for the Sumo Logic HTTP Logs and Metrics Source where Sensu should transmit the observability metrics. You can also provide the URL as a secret.
required	true

type	String

example without secrets

YML

```
url:
https://endpoint5.collection.us2.sumologic.com/receiver/v1/
http/xxxxxxxx
```

JSON

```
{
  "url":
"https://endpoint5.collection.us2.sumologic.com/receiver/v1
/http/xxxxxxxx"
}
```

example with secrets

YML

```
url: $SUMO_LOGIC_SOURCE_URL
```

JSON

```
{
  "url": "$SUMO_LOGIC_SOURCE_URL"
}
```

secrets

description	Array of the name/secret pairs to use with command execution. Read secrets attributes for details. You can also provide the Sumo Logic HTTP Logs and Metrics Source URL directly in the url attribute instead of configuring a secret.
required	false
type	String

YML

example

```yaml
secrets:
- name: SUMO_LOGIC_SOURCE_URL
  secret: sumologic_metrics_us2
```

JSON

```json
{
  "secrets": [
    {
      "name": "SUMOLOGIC_METRICS_URL",
      "secret": "sumologic_metrics_us2"
    }
  ]
}
```

max_connections

description	Maximum number of connections to keep alive in the connection pool. If set to `0`, there is no limit to the number of connections in the pool.
required	false
type	Integer

YML

example

```yaml
max_connections: 10
```

JSON

```json
{
  "max_connections": 10
}
```

timeout	
description	Duration to allow for processing a Sumo Logic call. In seconds.
required	true
type	String
example	**YML** `timeout: 10s` **JSON** `{ "timeout": "10s" }`

Secrets attributes

name	
description	Name of the secret defined in the handler's URL attribute. Becomes the environment variable presented to the handler. Read Use secrets management in Sensu for more information.
required	true
type	String
example	**YML** `name: SUMOLOGIC_METRICS_URL` **JSON** `{ "name": "SUMOLOGIC_METRICS_URL" }`

secret	
description	Name of the Sensu secret resource that defines how to retrieve the secret.
required	true
type	String
example	**YML** `secret: sumologic_metrics_us2` **JSON** `{ "secret": "sumologic_metrics_us2" }`

TCP stream handlers reference

***COMMERCIAL FEATURE**: Access TCP stream handlers in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu executes TCP stream handlers during the **process** stage of the observability pipeline.

Like traditional TCP handlers, TCP stream handlers send observability event data to TCP sockets for external services to consume. However, TCP stream handlers can help prevent the data bottlenecks you may experience with traditional TCP handlers.

Traditional TCP handlers start a new UNIX process for every Sensu event they receive and require a new connection to send every event. As you scale up and process more events per second, the rate at which the TCP handler can transmit observability event data decreases.

TCP stream handlers allow you to configure a connection pool with a maximum number of connections for the handler to use. For example, suppose you configure a TCP stream handler with a pool of 10 connections, and 1000 events are queued for transmission. As each connection finishes transmitting an event, it becomes available again and returns to the pool so the handler can use it to send the next event in the queue.

TCP stream handlers will reuse the available connections as long as they can rather than requiring a new connection for every event, which increases event throughput. In addition to providing a persistent TCP connection to transmit Sensu observation events to a remote data storage service, TCP stream handlers allow you to use transport layer security (TLS) for secure data transmission.

TCP stream handlers are commercial resources available for use in pipeline definitions.

TCP stream handler example

This example shows a TCP stream handler resource definition configured to use TLS:

YML

```
---
type: TCPStreamHandler
```

```
api_version: pipeline/v1
metadata:
  name: logstash
spec:
  address: 127.0.0.1:4242
  tls_ca_cert_file: "/path/to/tls/ca.pem"
  tls_cert_file: "/path/to/tls/cert.pem"
  tls_key_file: "/path/to/tls/key.pem"
  max_connections: 10
  min_reconnect_delay: 10ms
  max_reconnect_delay: 10s
```

JSON

```
{
  "type": "TCPStreamHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "logstash"
  },
  "spec": {
    "address": "127.0.0.1:4242",
    "tls_ca_cert_file": "/path/to/tls/ca.pem",
    "tls_cert_file": "/path/to/tls/cert.pem",
    "tls_key_file": "/path/to/tls/key.pem",
    "max_connections": 10,
    "min_reconnect_delay": "10ms",
    "max_reconnect_delay": "10s"
  }
}
```

Use TCP stream handlers

TCP stream handlers are commercial resources and are available for use **only** in pipelines.

> ***NOTE**: TCP stream handlers **are not** used by listing the handler name in the check handlers attribute.*

To use a TCP stream handler, list it as the handler in a pipeline definition. For example, this pipeline definition uses the logstash example along with the built-in is_incident event filter:

YML

```
---
type: Pipeline
api_version: core/v2
metadata:
  name: tcp_logging_workflows
spec:
  workflows:
  - name: log_all_incidents
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: logstash
      type: TCPStreamHandler
      api_version: pipeline/v1
```

JSON

```
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "tcp_logging_workflows"
  },
  "spec": {
    "workflows": [
      {
        "name": "log_all_incidents",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
```

```
            "name": "logstash",
            "type": "TCPStreamHandler",
            "api_version": "pipeline/v1"
          }
        }
      ]
    }
}
```

TCP stream handler specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. TCP stream handlers should always be type `TCPStreamHandler`.
required	Required for TCP stream handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: TCPStreamHandler` **JSON** `{ "type": "TCPStreamHandler" }`

api_version

description	Top-level attribute that specifies the Sensu API group and version. For TCP stream handlers in this version of Sensu, the `api_version` should always be `pipeline/v1`.
required	Required for TCP stream handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

```
api_version: pipeline/v1
```

JSON

```
{
  "api_version": "pipeline/v1"
}
```

metadata

description	Top-level collection of metadata about the TCP stream handler that includes `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the handler definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for TCP stream handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

```
metadata:
  name: logstash
  namespace: default
  created_by: admin
  labels:
    environment: development
```

```
    region: us-west-2
  annotations:
    managed-by: ops
```

JSON

```
{
  "metadata": {
    "name": "logstash",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "environment": "development",
      "region": "us-west-2"
    },
    "annotations": {
      "managed-by": "ops"
    }
  }
}
```

spec	
description	Top-level map that includes the TCP stream handler spec attributes.
required	Required for TCP stream handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  address: 127.0.0.1:4242` `  tls_ca_cert_file: "/path/to/tls/ca.pem"` `  tls_cert_file: "/path/to/tls/cert.pem"` `  tls_key_file: "/path/to/tls/key.pem"` `  max_connections: 10` `  min_reconnect_delay: 10ms`

```
  max_reconnect_delay: 10s
```

JSON

```
{
  "spec": {
    "address": "127.0.0.1:4242",
    "tls_ca_cert_file": "/path/to/tls/ca.pem",
    "tls_cert_file": "/path/to/tls/cert.pem",
    "tls_key_file": "/path/to/tls/key.pem",
    "max_connections": 10,
    "min_reconnect_delay": "10ms",
    "max_reconnect_delay": "10s"
  }
}
```

Metadata attributes

name	
description	Unique string used to identify the TCP stream handler. TCP stream handler names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each TCP stream handler must have a unique name within its namespace.
required	true
type	String
example	**YML** `name: logstash` **JSON** `{ "name": "logstash" }`

namespace

description	Sensu RBAC namespace that the TCP stream handler belongs to.
required	false
type	String
default	`default`
example	**YML** `namespace: default` **JSON** `{ "namespace": "default" }`

created_by

description	Username of the Sensu user who created the TCP stream handler or last updated the TCP stream handler. Sensu automatically populates the `created_by` field when the TCP stream handler is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON**

```
{
  "created_by": "admin"
}
```

labels	
description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

YML

example

```
labels:
  environment: development
  region: us-west-2
```

JSON

```
{
  "labels": {
    "environment": "development",
    "region": "us-west-2"
  }
}
```

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` **JSON** `{` `  "annotations": {` `  "managed-by": "ops"` `  }` `}`

Spec attributes

address	
description	The hostname:port combination the TCP stream handler should connect to.

required	true
type	String

example

YML

```yaml
address: 127.0.0.1:4242
```

JSON

```json
{
  "address": "127.0.0.1:4242"
}
```

tls_ca_cert_file

description	Path to the PEM-format CA certificate to use for TLS client authentication.
required	false
type	String

example

YML

```yaml
tls_ca_cert_file: "/path/to/tls/ca.pem"
```

JSON

```json
{
  "tls_ca_cert_file": "/path/to/tls/ca.pem"
}
```

tls_cert_file

description	Path to the PEM-format certificate to use for TLS client authentication.

	This certificate and its corresponding key are required for secure client communication.
required	false
type	String
example	**YML** `tls_cert_file: "/path/to/tls/cert.pem"` **JSON** `{ "tls_cert_file": "/path/to/tls/cert.pem" }`

tls_key_file

description	Path to the PEM-format key file associated with the tls_cert_file to use for TLS client authentication. This key and its corresponding certificate are required for secure client communication.
required	false
type	String
example	**YML** `tls_key_file: "/path/to/tls/key.pem"` **JSON** `{ "tls_key_file": "/path/to/tls/key.pem" }`

max_connections

description	Maximum number of connections to keep alive in the connection pool. If set to `0`, connection pooling is disabled.
required	true
type	Integer
example	**YML** `max_connections: 10` **JSON** `{ "max_connections": 10 }`

max_reconnect_delay

description	Maximum time to wait while retrying a broken connection. In seconds (`s`) or milliseconds (`ms`).
required	true
type	String
example	**YML** `max_reconnect_delay: 10s` **JSON** `{ "max_reconnect_delay": "10s" }`

min_reconnect_delay

description	Minimum time to wait while retrying a broken connection. In seconds (`s`) or milliseconds (`ms`).
required	true
type	String
example	**YML** `min_reconnect_delay: 10ms` **JSON** `{ "min_reconnect_delay": "10ms" }`

Aggregate metrics with the Sensu StatsD listener

StatsD is a daemon, tool, and protocol that you can use to send, collect, and aggregate custom metrics Services that implement StatsD typically expose UDP port 8125 to receive metrics according to the line protocol `<metricname>:<value>|<type>`.

With StatsD, you can measure anything and everything. Collect custom metrics in your code and send them to a StatsD server to monitor applicaton performance. Monitor CPU, I/O, and network system levels with collection daemons. You can feed the metrics that StatsD aggregates to multiple different backends to store or visualize the data.

Use Sensu to implement StatsD

Sensu implements a StatsD listener on its agents. Each `sensu-agent` listens on the default port 8125 for UDP messages that follow the StatsD line protocol. StatsD aggregates the metrics, and Sensu translates them to Sensu metrics and events that can be passed to the event pipeline. You can configure the StatsD listener and access it with the netcat utility command:

```
echo 'abc.def.g:10|c' | nc -w1 -u localhost 8125
```

Metrics received through the StatsD listener are not stored in etcd. Instead, you must configure event handlers to send the data to a storage solution (for example, a time-series database like InfluxDB).

Configure the StatsD listener

Use configuration flags to configure the Sensu StatsD Server when you start up a `sensu-agent`.

The following flags allow you to configure event handlers, flush interval, address, and port:

```
--statsd-disable                    disables the statsd listener and metrics
server
```

```
--statsd-event-handlers stringSlice   comma-delimited list of event handlers for
statsd metrics
--statsd-flush-interval int           number of seconds between statsd flush (default
10)
--statsd-metrics-host string          address used for the statsd metrics server
(default "127.0.0.1")
--statsd-metrics-port int             port used for the statsd metrics server
(default 8125)
```

For example:

```
sensu-agent start --statsd-event-handlers influx-db --statsd-flush-interval 1 --
statsd-metrics-host "123.4.5.6" --statsd-metrics-port 8125
```

Next steps

Now that you know how to feed StatsD metrics into Sensu, check out these resources to learn how to handle the StatsD metrics:

- Handlers reference: in-depth documentation for Sensu handlers
- InfluxDB handler guide: instructions on Sensu's built-in metric handler
- Pipelines reference: information about the Sensu pipeline resource, which you can use to create event processing workflows with event filters, mutators, and handlers

Create handler templates

Sensu Go uses the Go template package, which allows you to generate text output that includes observation data from events. Sensu handler templates include HTML-formatted text and data derived from event attributes like `event.entity.name` and `event.check.output`. This allows you to add meaningful, actionable context to alerts.

For example, a template for a brief Slack alert might include information about the affected entity and its status, as well as a link to the organization's playbook for resolving observability alerts:

```
<html>
The entity {{.Entity.Name}} has a status of {{.Check.State}}. The entity has
reported the same status for {{.Check.Occurrences}} preceding events.<br>
The playbook for managing this alert is available at
https://example.com/observability/alerts/playbook.
</html>
```

Template syntax and format

Handler templates use dot notation syntax to access event attributes, with the event attribute wrapped in double curly braces. The initial dot indicates `event`.

For example, in a handler template, a reference to the event attribute `event.check.occurrences` becomes `{{.Check.Occurrences}}`.

Use HTML to format the text and spacing in your templates. All text outside double curly braces is copied directly into the template output, with HTML formatting applied.

Available event attributes

If you are using a plugin that supports template output, every attribute in the Sensu event is available. However, the attribute capitalization pattern is different for handler templates than for event format.

The table below lists the event attributes that are available to use in handler templates, in the correct dot notation and capitalization pattern. You can also use the template toolkit command to print available event attributes for a specific event.

***NOTE**: The entity and events specifications describe each attribute in detail.*

attribute	attribute	attribute
`.HasCheck`	`.HasMetrics`	`.IsIncident`
`.IsResolution`	`.IsSilenced`	`.Timestamp`
`.Check.Annotations`	`.Check.CheckHooks`	`.Check.Command`
`.Check.Cron`	`.Check.DiscardOutput`	`.Check.Duration`
`.Check.EnvVars`	`.Check.Executed`	`.Check.ExtendedAttributes`
`.Check.Handlers`	`.Check.HighFlapThreshold`	`.Check.History`
`.Check.Hooks`	`.Check.Interval`	`.Check.Issued`
`.Check.Labels`	`.Check.LastOK`	`.Check.LowFlapThreshold`
`.Check.MaxOutputSize`	`.Check.Name`	`.Check.Namespace`
`.Check.Occurrences`	`.Check.OccurrencesWatermark`	`.Check.Output`
`.Check.OutputMetricFormat`	`.Check.OutputMetricHandlers`	`.Check.ProxyEntityName`
`.Check.ProxyRequests`	`.Check.Publish`	`.Check.RoundRobin`
`.Check.RuntimeAssets`	`.Check.Secrets`	`.Check.Silenced`
`.Check.State`	`.Check.Status`	`.Check.Stdin`

`.Check.Subdue`	`.Check.Subscriptions`	`.Check.Timeout`
`.Check.TotalStateChange`	`.Check.Ttl`	`.Entity.Annotations`
`.Entity.Deregister`	`.Entity.Deregistration`	`.Entity.EntityClass`
`.Entity.ExtendedAttributes`	`.Entity.KeepaliveHandlers`	`.Entity.Labels`
`.Entity.LastSeen`	`.Entity.Name`	`.Entity.Namespace`
`.Entity.Redact`	`.Entity.SensuAgentVersion`	`.Entity.Subscriptions`
`.Entity.System`	`.Entity.System.Arch`	`.Entity.System.ARMVersion`
`.Entity.System.CloudProvider`	`.Entity.System.Hostname`	`.Entity.System.LibcType`
`.Entity.System.Network`	`.Entity.System.OS`	`.Entity.System.Platform`
`.Entity.System.PlatformFamily`	`.Entity.System.PlatformVersion`	`.Entity.System.Processes`
`.Entity.System.VMRole`	`.Entity.System.VMSystem`	`.Entity.User`
`.Metrics.Handlers`	`.Metrics.Points`	

Template toolkit command

The sensu/template-toolkit-command dynamic runtime asset provides a sensuctl command plugin you can use to print a list of available event attributes in handler template dot notation syntax and validate your handler template output.

The template toolkit command uses event data you supply via stdin in JSON format.

Add the Sensu template toolkit command asset to Sensu:

```
sensuctl asset add sensu/template-toolkit-command:0.4.0 -r template-toolkit-command
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `template-toolkit-command`.

You can also download the latest asset definition from Bonsai.

Run `sensuctl asset list` to confirm that the asset is ready to use:

```
          Name                                              URL                                        Hash
 ─────────────────────────── ─────────────────────────────────────────────────────────────────────── ─────────
  template-toolkit-command   //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_windows_amd64.tar.gz   019ccf3
  template-toolkit-command   //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_darwin_amd64.tar.gz    b771813
  template-toolkit-command   //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_armv7.tar.gz     4e7ad65
  template-toolkit-command   //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_arm64.tar.gz     02eca1f
  template-toolkit-command   //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_386.tar.gz       56ed603
  template-toolkit-command   //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_amd64.tar.gz     7dbd2c6
```

Print available event attributes

Use the template toolkit command to print a list of the available event attributes as well as the correct dot notation and capitalization pattern for a specific event (in this example, `event.json`):

```
cat event.json | sensuctl command exec template-toolkit-command -- --dump-names
```

The response lists the available attributes for the event:

```
INFO[0000] asset includes builds, using builds instead of asset  asset=template-
toolkit-command component=asset-manager entity=sensuctl
.Event{
    .Timestamp: 1580310179,
 .Entity{
     .EntityClass: "agent",
     .System:      .System{
 [...]
 .Check{
     .Command:           "",
     .Handlers:          {"keepalive"},
     .HighFlapThreshold: 0x0,
 [...]
```

In this example, the response lists the available event attributes `.Timestamp`, `.Entity.EntityClass`, `.Entity.System`, `.Check.Command`, `.Check.Handlers`, and `.Check.HighFlapThreshold`.

You can also use `sensuctl event info <entity_name> <check_name>` to print the correct notation and pattern: template output for a specific event (in this example, an event for entity `server01` and check `server-health`):

```
sensuctl event info server01 server-health --format json | sensuctl command exec
template-toolkit -- --dump-names
```

The response lists the available attributes for the event:

```
INFO[0000] asset includes builds, using builds instead of asset  asset=template-
toolkit-command component=asset-manager entity=sensuctl
.Event{
    .Timestamp: 1580310179,
    .Entity:{
        .EntityClass:        "proxy",
        .System:             .System{
 [...]
    .Check:{
        .Command:           "health.sh",
        .Handlers:          {"slack"},
```

```
        .HighFlapThreshold: 0x0,
    [...]
```

Validate handler template output

Use the template toolkit command to validate the dot notation syntax and output for any event attribute.

For example, to test the output for the `{{.Check.Name}}` attribute for the event `event.json`:

```
cat event.json | sensuctl command exec template-toolkit-command -- --template "
{{.Check.Name}}"
```

The response will list the template output:

```
INFO[0000] asset includes builds, using builds instead of asset  asset=template-
toolkit-command component=asset-manager entity=sensuctl
executing command with --template {{.Check.Name}}
Template String Output: keepalive
```

In this example, the command validates that for the `event.json` event, the handler template will replace `{{.Check.Name}}` with `keepalive` in template output.

You can also use `sensuctl event info <entity_name> <check_name>` to validate template output for a specific event (in this example, an event for entity `webserver01` and check `check-http`):

```
sensuctl event info webserver01 check-http --format json | sensuctl command exec
template-toolkit-command -- --template "Server: {{.Entity.Name}} Check:
{{.Check.Name}} Status: {{.Check.State}}"
```

The response will list the template output:

```
Executing command with --template Server: {{.Entity.Name}} Check: {{.Check.Name}}
Status: {{.Check.State}}
Template String Output: Server: "webserver01 Check: check-http Status: passing"
```

Sensu Email Handler plugin

The Sensu Email Handler plugin allows you to provide a template for the body of the email. For example, this template will produce an email body that includes the name of the check and entity associated with the event, the status and number of occurrences, and other event details:

```
<html>
Greetings,

<h3>Informational Details</h3>
<b>Check</b>: {{.Check.Name}}<br>
<b>Entity</b>: {{.Entity.Name}}<br>
<b>State</b>: {{.Check.State}}<br>
<b>Occurrences</b>: {{.Check.Occurrences}}<br>
<b>Playbook</b>: https://example.com/monitoring/wiki/playbook
<h3>Check Output Details</h3>
<b>Check Output</b>: {{.Check.Output}}
<h4>Check Hook(s)</h4>
{{range .Check.Hooks}}<b>Hook Name</b>:  {{.Name}}<br>
<b>Hook Command</b>:  {{.Command}}<br>
<b>Hook Output</b>: {{.Output}}<br>
{{end}}<br>
<br>
<br>
#monitoringlove,<br>
<br>
Sensu<br>
</html>
```

The sensu/sensu-email-handler dynamic runtime asset also includes a UnixTime function that allows you to print timestamp values from events in human-readable format. Read the sensu/sensu-email-handler Bonsai page for details.

Sensu PagerDuty Handler example

The sensu/sensu-pagerduty-handler dynamic runtime asset includes a basic template for the PagerDuty alert summary:

```
"{{.Entity.Name}}/{{.Check.Name}} : {{.Check.Output}}"
```

With this template, the summary for every alert in PagerDuty will include:

- The name of the affected entity.
- The name of the check that produced the event.
- The check output for the event.

Read the Sensu PagerDuty Handler Bonsai page for details.

Plan maintenance windows with silencing

As the Sensu backend processes check results, the server executes handlers to send alerts or otherwise relay observation events and metrics data to external services. Sensu's built-in silencing capability allows you to suppress event handler execution as needed. This feature is useful when you're planning maintenance.

You can configure silences to prevent handlers from taking actions based on check name, entity subscription, entity name, or a combination of these factors. In this guide, you'll create a silenced entry for a specific entity and its associated check to prevent alerts and create a time window for maintenance.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

> ***NOTE**: If you already have an entity and running check to use as the silencing target, skip ahead to Create the silenced entry.*

Configure a Sensu entity

Before you create a check, you'll need a Sensu entity with the subscription `website` to run the check. Use sensuctl to add the `website` subscription to an entity the Sensu agent is observing.

> ***NOTE**: To find your entity name, run `sensuctl entity list`. The `ID` is the name of your entity.*

Before you run the following code, replace `<ENTITY_NAME>` with the name of the entity on your system.

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `website` and press enter.

Before you continue, confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register the http-checks dynamic runtime asset

To power the check in your silenced entry, you'll use the sensu/http-checks dynamic runtime asset. This community-tier asset includes `http-check`, the http status check command that your check will rely on.

Register the sensu/http-checks dynamic runtime asset:

```
sensuctl asset add sensu/http-checks:0.5.0 -r http-checks
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `http-checks`. The response will indicate that the asset was added.

Use sensuctl to confirm that the dynamic runtime asset is ready to use:

```
sensuctl asset list
```

The response should list the sensu/http-checks dynamic runtime asset (renamed to `http-checks`):

Name	URL	Hash
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_windows_amd64.tar.gz	52ae075
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_darwin_amd64.tar.gz	72d0f15
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_armv7.tar.gz	ef18587
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_arm64.tar.gz	3504ddf
http-checks	//assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_386.tar.gz	60b8883

```
http-checks  //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_amd64.tar.gz  1db73a8
```

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Create the check

With the dynamic runtime asset registered, you can create a check named `check-website` to run the command `http-check --url https://sensu.io`, at an interval of 15 seconds, for all agents subscribed to the `website` subscription, using the `sensu-site` proxy entity name.

To add the check, run:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check-website
spec:
  command: http-check --url https://sensu.io
  interval: 15
  proxy_entity_name: sensu-site
  publish: true
  round_robin: true
  runtime_assets:
  - http-checks
  subscriptions:
  - website
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
```

```
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-website"
  },
  "spec": {
    "command": "http-check --url https://sensu.io",
    "interval": 15,
    "proxy_entity_name": "sensu-site",
    "publish": true,
    "round_robin": true,
    "runtime_assets": [
      "http-checks"
    ],
    "subscriptions": [
      "website"
    ]
  }
}
EOF
```

Use sensuctl to confirm that Sensu added the check:

```
sensuctl check list
```

The response should list `check-sensu-site`:

```
    Name          Command          Interval   Cron   Timeout   TTL   Subscriptions   Handlers   Assets   Hooks
Publish?   Stdin?   Metric Format   Metric Handlers
──────────────── ──────────────────────────────── ────────── ─────── ───────── ───── ─────────────── ────────── ──────── ──────
──────────── ──────── ──────────── ──────────────
 check-website   http-check --url https://sensu.io   15           0     0     website           http-checks          true       false
```

Create the silenced entry

The silenced entry will silence the check `check-http` on the entity `sensu-site` for a planned maintenance window that:

- Starts at **04:00** UTC on **March 14, 2022**
- Automatically ends **1 hour** later
- Adds your username as the **creator** of the silenced entry

To create the silenced entry, run:

```
sensuctl silenced create \
--subscription 'entity:sensu-site' \
--check 'check-http' \
--begin '2022-03-14 04:00:00 -00:00' \
--expire 3600 \
--reason 'Planned site maintenance'
```

***NOTE**: Sensuctl supports several time formats for the `begin` flag. This example uses RFC 3339 format with space delimiters and numeric zone offset.*

Use sensuctl to verify that the silenced entry against the entity `sensu-site` was created properly:

SHELL

```
sensuctl silenced info 'entity:sensu-site:check-http' --format yaml
```

SHELL

```
sensuctl silenced info 'entity:sensu-site:check-http' --format wrapped-json
```

The response will list the silenced resource definition, similar to the following:

YML

```
type: Silenced
api_version: core/v2
metadata:
```

```
  name: entity:sensu-site:check-http
spec:
  begin: 1647230400
  check: check-http
  creator: admin
  expire: 3600
  expire_at: 1647234000
  expire_on_resolve: false
  reason: Planned site maintenance
  subscription: entity:sensu-site
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "entity:sensu-site:check-http"
  },
  "spec": {
    "begin": 1647230400,
    "check": "check-http",
    "creator": "admin",
    "expire": 3600,
    "expire_at": 1647234000,
    "expire_on_resolve": false,
    "reason": "Planned site maintenance",
    "subscription": "entity:sensu-site"
  }
}
```

Next steps

When your silence goes into effect at the designated `begin` time, you will still see events for `check-http` on the `sensu-site` entity in the Sensu web UI. This is because **silences do not stop events from being produced — they stop events from being handled**.

If you followed this guide to create the `check-http` check on the `sensu-site` entity, you might have noticed that the check does not include a pipeline. To observe the silenced entry's effect, add a pipeline

to the `check-http` check definition (or recreate the silenced entry with your own entity and a check that includes a pipeline). The pipeline must include a workflow with the built-in `not_silenced` event filter and a handler.

> ***WARNING**: By default, silenced events will be handled unless the pipeline workflow includes the built-in `not_silenced` event filter to discard silenced events.*

Follow one of these guides to add a pipeline to your check:

- Send data to Sumo Logic with Sensu
- Send email alerts with a pipeline
- Send PagerDuty alerts with Sensu
- Send Slack alerts with a pipeline

Read the silencing reference for in-depth documentation about silenced entries and more examples:

- Silence all checks on a specific entity
- Silence a specific check on a specific entity
- Silence all checks with a specific subscription
- Silence all checks for entities with a specific subscription
- Silence a specific check on entities with a specific subscription
- Silence a specific check on every entity

Populate metrics in InfluxDB with handlers

> ***PRO TIP**: You can use the InfluxDB Metrics integration in the Sensu Catalog to send Sensu event data to InfluxDB instead of following this guide. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

A Sensu event handler is an action the Sensu backend executes when a specific event occurs. In this guide, you'll use a handler to populate the time-series database InfluxDB with Sensu observability event data.

Metrics can be collected from check output (in this guide, a check that generates Prometheus metrics) or the Sensu StatsD Server.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition.

The example in this guide uses the `prometheus_metrics` check from Collect Prometheus metrics with Sensu, which includes the subscription `app_tier`. Use sensuctl to add an `app_tier` subscription to one of your entities.

Before you run the following code, replace `<ENTITY_NAME>` with the name of the entity on your system.

> ***NOTE**: To find your entity name, run `sensuctl entity list`. The `ID` is the name of your entity.*

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `app_tier` and press enter.

Run this command to confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register the dynamic runtime asset

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. This example uses the sensu/sensu-influxdb-handler dynamic runtime asset to power an InfluxDB handler.

Use `sensuctl asset add` to register the sensu/sensu-influxdb-handler dynamic runtime asset:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.7.0 -r sensu-influxdb-handler
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-influxdb-handler:3.7.0 -r sensu-influxdb-handler
added asset: sensu/sensu-influxdb-handler:3.7.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-influxdb-handler"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `sensu-influxdb-handler`.

You can also download the latest dynamic runtime asset definition for your platform from Bonsai and

register the asset with `sensuctl create --file filename.yml` or `sensuctl create --file filename.json`.

Run `sensuctl asset list` to confirm that the dynamic runtime asset is ready to use.

> ***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Create the handler

Now that you have registered the dynamic runtime asset, use sensuctl to create a handler called `influxdb-handler` that pipes observation data (events) to InfluxDB with the sensu/sensu-influxdb-handler dynamic runtime asset.Edit the command below to replace the placeholders for database name, address, username, and password with the information for your own InfluxDB database.For more information about the Sensu InfluxDB handler, read the asset page in Bonsai.

```
sensuctl handler create influxdb-handler \
--type pipe \
--command "sensu-influxdb-handler -d sensu" \
--env-vars "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086,
INFLUXDB_USER=sensu, INFLUXDB_PASS=password" \
--runtime-assets sensu-influxdb-handler
```

You should receive a confirmation message:

```
Created
```

To review the complete resource definition for the handler resource you just created with sensuctl, run:

SHELL

```
sensuctl handler info influxdb-handler --format yaml
```

SHELL

```
sensuctl handler info influxdb-handler --format wrapped-json
```

The `influxdb-handler` resource definition will be similar to this example:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: influxdb-handler
spec:
  command: sensu-influxdb-handler -d sensu
  env_vars:
  - INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086
  - INFLUXDB_USER=sensu
  - INFLUXDB_PASS=password
  filters: null
  handlers: null
  runtime_assets:
  - sensu-influxdb-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "influxdb-handler"
  },
  "spec": {
    "command": "sensu-influxdb-handler -d sensu",
    "env_vars": [
      "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
      "INFLUXDB_USER=sensu",
      "INFLUXDB_PASS=password"
    ],
```

```
    "filters": null,
    "handlers": null,
    "runtime_assets": [
      "sensu-influxdb-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

You can share, reuse, and maintain this handler just like you would code: save it to a file and start building a monitoring as code repository.

> ***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Create a pipeline that includes the InfluxDB handler

With your handler configured, you can add it to a pipeline workflow. A single pipeline workflow can include one or more filters, one mutator, and one handler.

In this case, the pipeline includes the built-in has_metrics and not_silenced event filters and the InfluxDB handler you've already configured. To create the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: metrics_pipeline
spec:
  workflows:
  - name: influxdb_metrics
    filters:
    - name: has_metrics
      type: EventFilter
```

```
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    handler:
      name: influxdb-handler
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "metrics_pipeline"
  },
  "spec": {
    "workflows": [
      {
        "name": "influxdb_metrics",
        "filters": [
          {
            "name": "has_metrics",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "influxdb-handler",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
```

```
      ]
    }
  }
  EOF
```

Now you can add the `metrics_pipeline` pipeline to a check for check output metric extraction.

Add the pipeline to a check

Add the `metrics_pipeline` pipeline to a check to use it for check output metric extraction.For example, if you followed Collect Prometheus metrics with Sensu, you created the `prometheus_metrics` check.

The `prometheus_metrics` check already uses the `influxdb_line` output metric format, but you will need to add the pipeline to extract the metrics and process them according to the pipeline's workflows.

To open the check definition in your text editor, run:

```
sensuctl edit check prometheus_metrics
```

Make two changes in the `prometheus_metrics` check definition:

1. Delete the `output_metrics_handlers` attribute and value.
2. Replace the `pipelines: []` line with the following array to reference your `metrics_pipeline` pipeline:

   YML

   ```
   pipelines:
   - type: Pipeline
     api_version: core/v2
     name: metrics_pipeline
   ```

   JSON

   ```
   {
   ```

```
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "metrics_pipeline"
    }
  ]
}
```

Save the two changes and exit the text editor. You should receive a confirmation message:

```
Updated /api/core/v2/namespaces/default/checks/prometheus_metrics
```

To review the updated check resource definition, run:

SHELL

```
sensuctl check info prometheus_metrics --format yaml
```

SHELL

```
sensuctl check info prometheus_metrics --format wrapped-json
```

The updated `prometheus_metrics` check definition will be similar to this example:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: prometheus_metrics
spec:
  check_hooks: null
  command: sensu-prometheus-collector -prom-url http://localhost:9090 -prom-query up
  env_vars: null
  handlers: []
```

```
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: influxdb_line
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: metrics_pipeline
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu-prometheus-collector
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - app_tier
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "prometheus_metrics"
  },
  "spec": {
    "check_hooks": null,
    "command": "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
query up",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "influxdb_line",
    "output_metric_handlers": null,
    "pipelines": [
```

```
      {
        "api_version": "core/v2",
        "name": "metrics_pipeline",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu-prometheus-collector"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "app_tier"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Assign the InfluxDB handler to the Sensu StatsD listener

To assign your `influxdb-handler` resource to the Sensu StatsD listener at agent startup and pass all StatsD metrics into InfluxDB:

```
sensu-agent start --statsd-event-handlers influxdb-handler
```

Validate the InfluxDB handler

It might take a few moments for Sensu to receive metrics after you assign the pipeline to the check or assign the handler to the StatsD server. After an event is handled, metrics should start populating InfluxDB. You can verify proper handler behavior with `sensu-backend` logs. Read Troubleshoot Sensu for log locations by platform.

Whenever an event is being handled, a log entry is added with the message `"handler":"influxdb-handler","level":"debug","msg":"sending event to handler"`, followed by a second log entry with the message `"msg":"pipelined executed event pipe handler","output":"","status":0`.

Next steps

Now that you know how to apply an InfluxDB handler to metrics, read Aggregate metrics with the Sensu StatsD listener to learn more about using Sensu to implement StatsD and take action on observability events.

Send data to Sumo Logic with Sensu

> ***PRO TIP**: You can use the Sumo Logic Analytics integration in the Sensu Catalog to send Sensu event data to Sumo Logic instead of following this guide. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Follow this guide to create a pipeline that sends data from a Sensu check to Sumo Logic for long-term logs and metrics storage. Sensu checks are commands the Sensu agent executes that generate observability data in a status or metric event. Sensu pipelines define the event filters and actions the Sensu backend executes on the events.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

In addition, this guide uses an example check named `check_cpu`. If you don't already have this check in place, follow Monitor server resources to add it.

Configure a Sensu entity

Sensu checks have a subscriptions attribute, where you specify strings to indicate which subscribers will execute the checks. For Sensu to execute a check, at least one entity must include a subscription that matches a subscription in the check definition. In the example in this guide, the `check_cpu` check includes the `system` subscription, so at least one entity must subscribe to `system` to run the check.

First, select the entity whose data you want to send to Sumo Logic. To list all of your entities in the current namespace, run:

```
sensuctl entity list
```

The `ID` in the response is the entity name. Select one of the listed entities.

Before you run the following sensuctl command, replace `<ENTITY_NAME>` with the name of your entity. Then run the command to add the `system` subscription to your entity:

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `system` and press enter.

Finally, confirm that both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register the dynamic runtime asset

The sensu/sensu-sumologic-handler dynamic runtime asset includes the scripts your handler will need to send observability data to Sumo Logic.

To add the sensu/sensu-sumologic-handler asset, run:

```
sensuctl asset add sensu/sensu-sumologic-handler:0.3.0 -r sumologic-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `sumologic-handler`.

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-sumologic-handler:0.3.0
added asset: sensu/sensu-sumologic-handler:0.3.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sumologic-handler"].
```

To confirm that the asset was added to your Sensu backend, run:

```
sensuctl asset info sumologic-handler
```

The response will list the available builds for the sensu/sensu-sumologic-handler dynamic runtime asset.

> ***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read the asset reference for more information about dynamic runtime asset builds.*

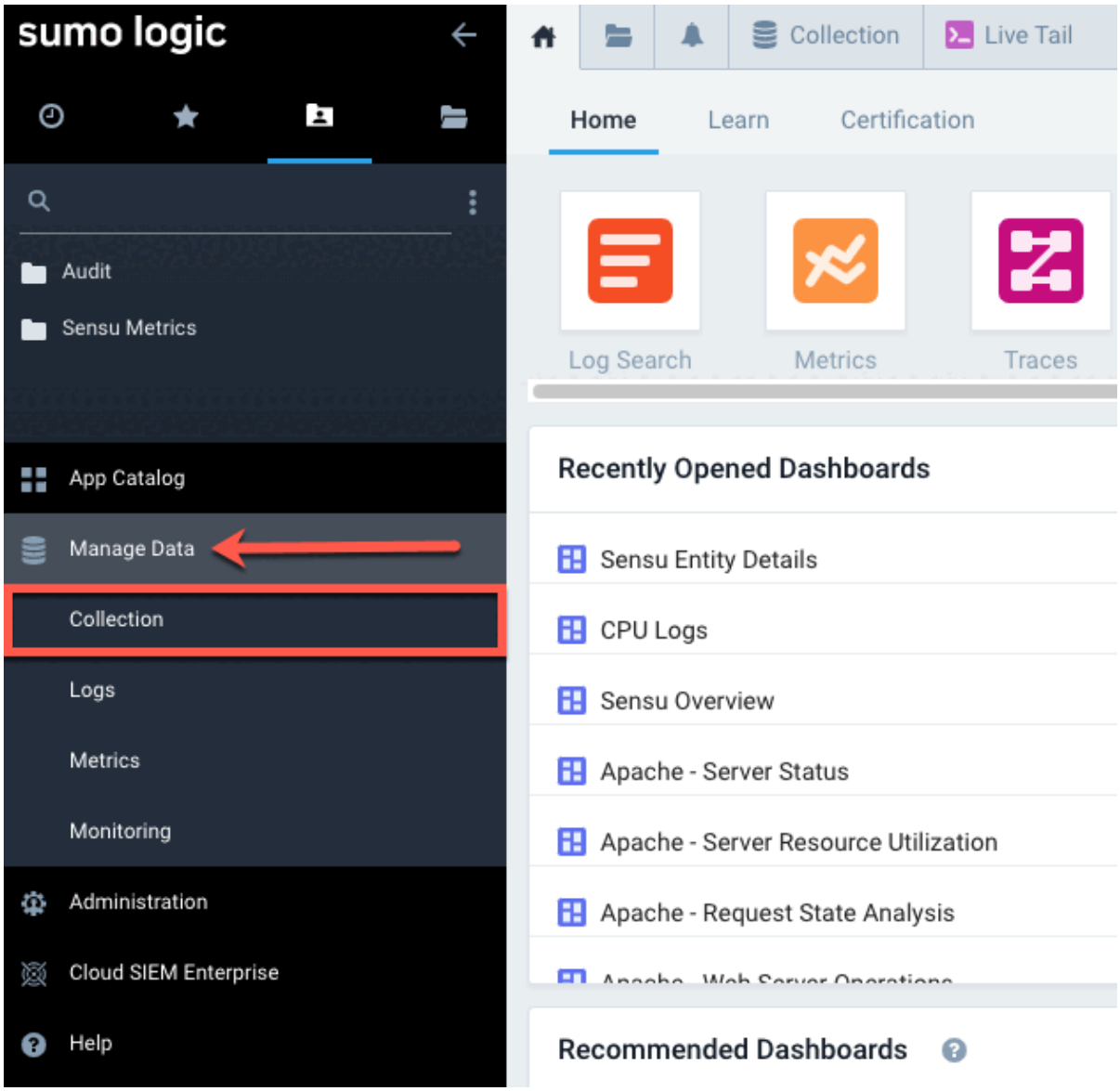
Set up an HTTP Logs and Metrics Source

Set up a Sumo Logic HTTP Logs and Metrics Source to collect your Sensu observability data.

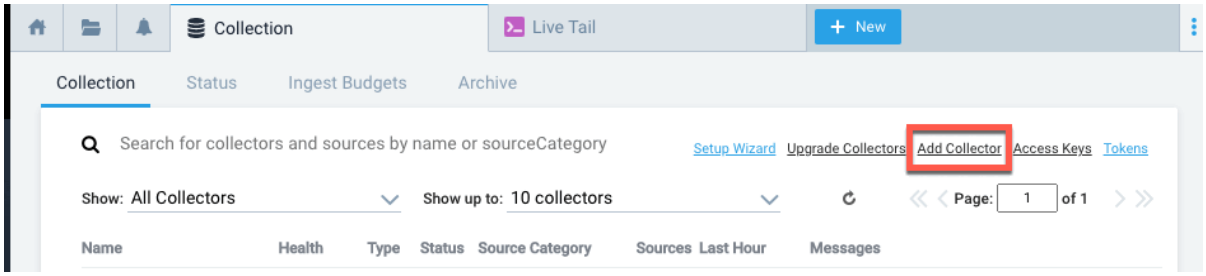
> ***NOTE**: If you have an existing Sumo Logic HTTP Logs and Metrics Source, you can send Sensu data there instead of creating a new source if you wish. Copy the HTTP Source Address URL for your existing source and skip to Add the Sumo Logic handler.*

Log in to your Sumo Logic account and follow these instructions:

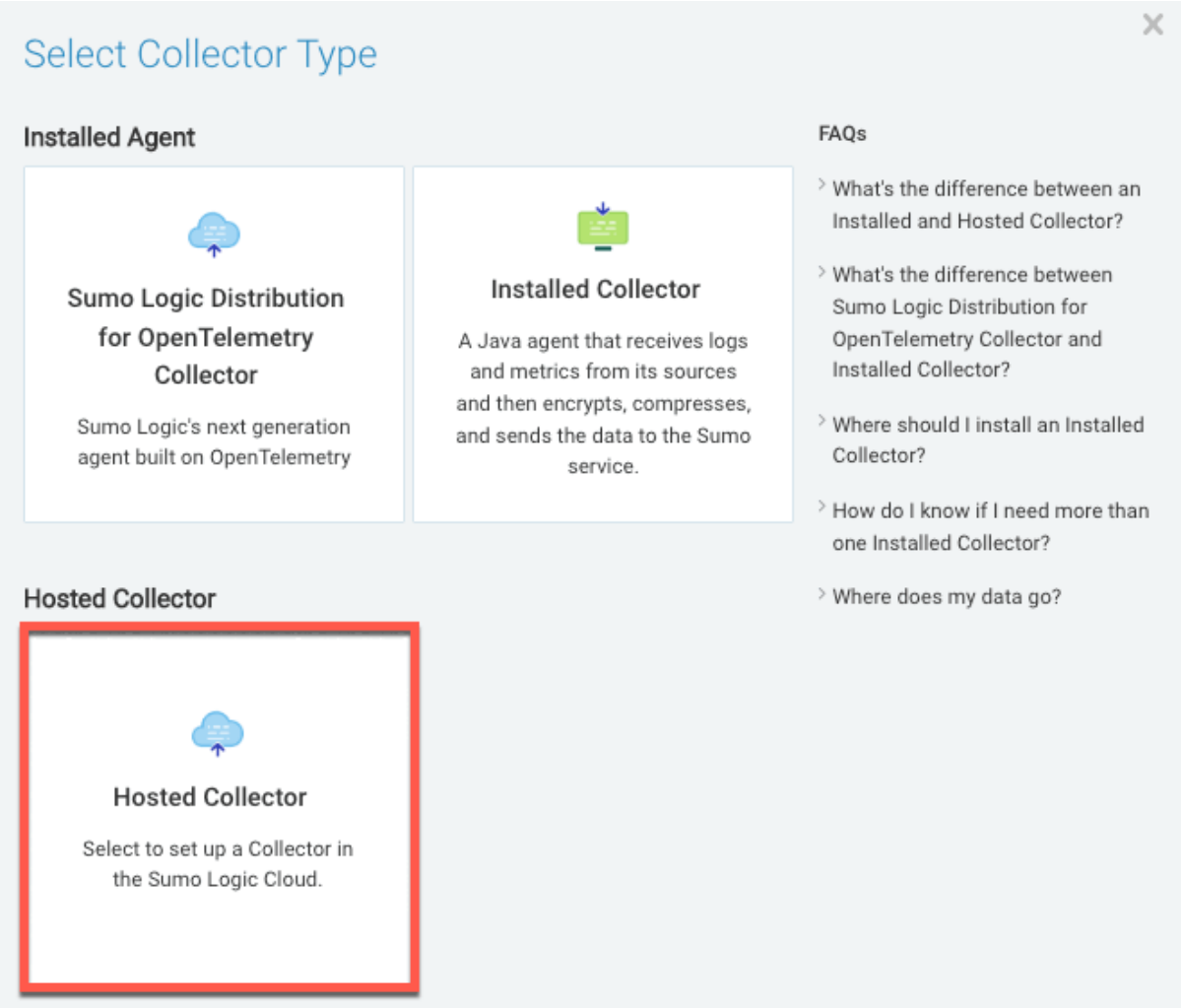
1. In the Sumo Logic left-navigation menu, click **Manage Data** and then **Collection** to open the Collection tab.



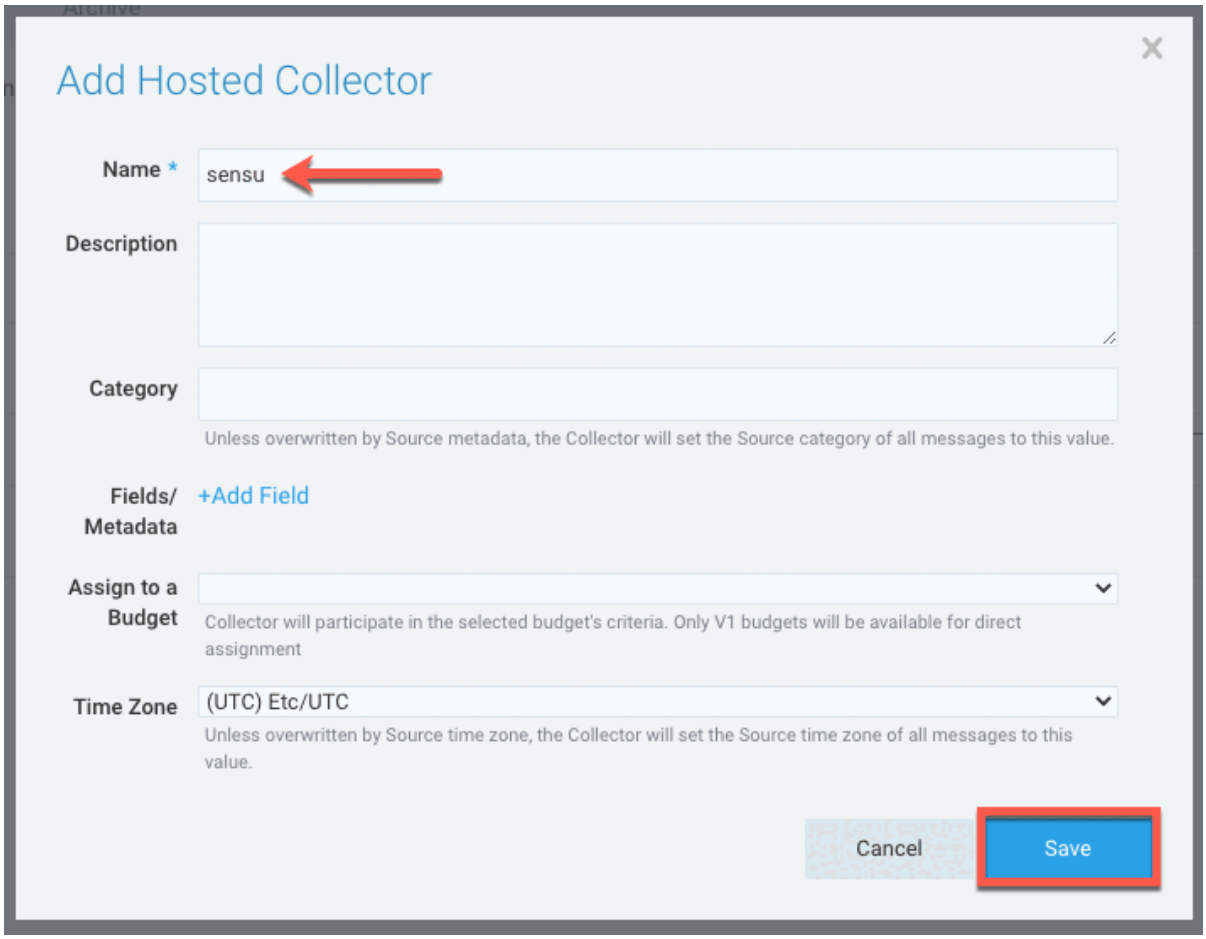
2. At the top-right of the Collection tab, click **Add Collector**.



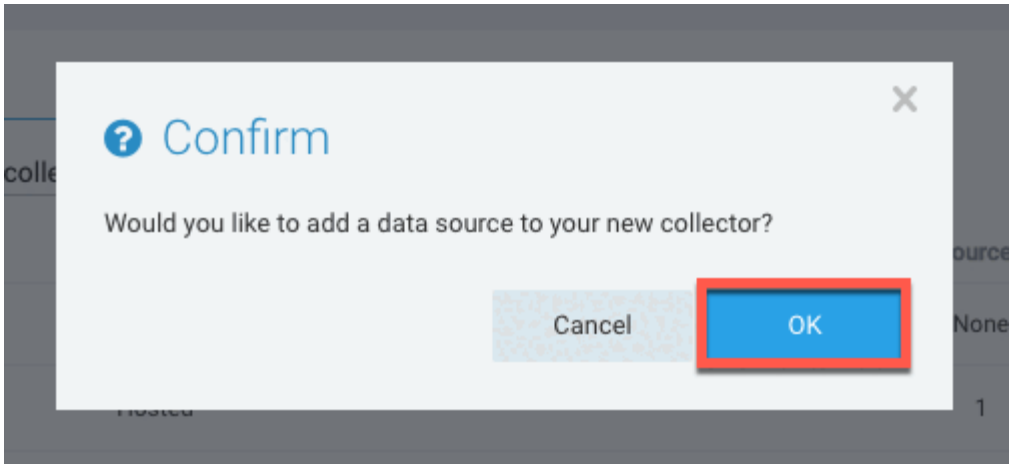
3. In the Click Selector Type modal window, click **Hosted Collector**.



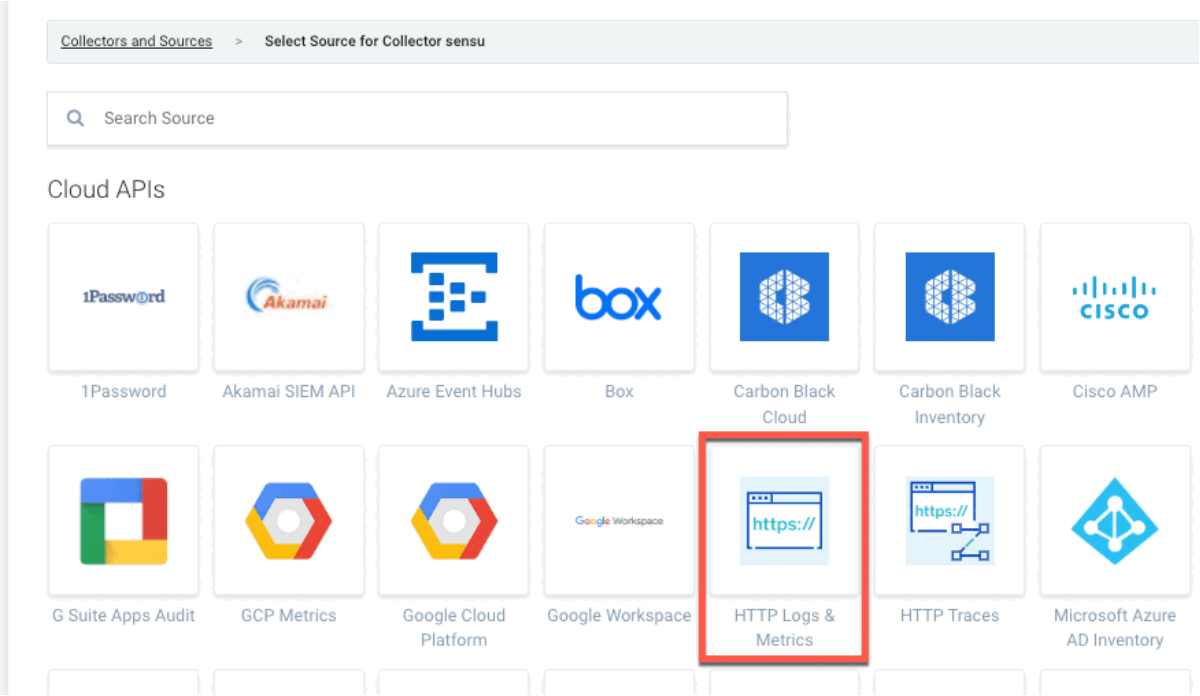
4. In the Add Hosted Collector modal window:
    - Type **sensu** in the Name field.
    - Click **Save**.



5. In the Confirm prompt, click **OK**.

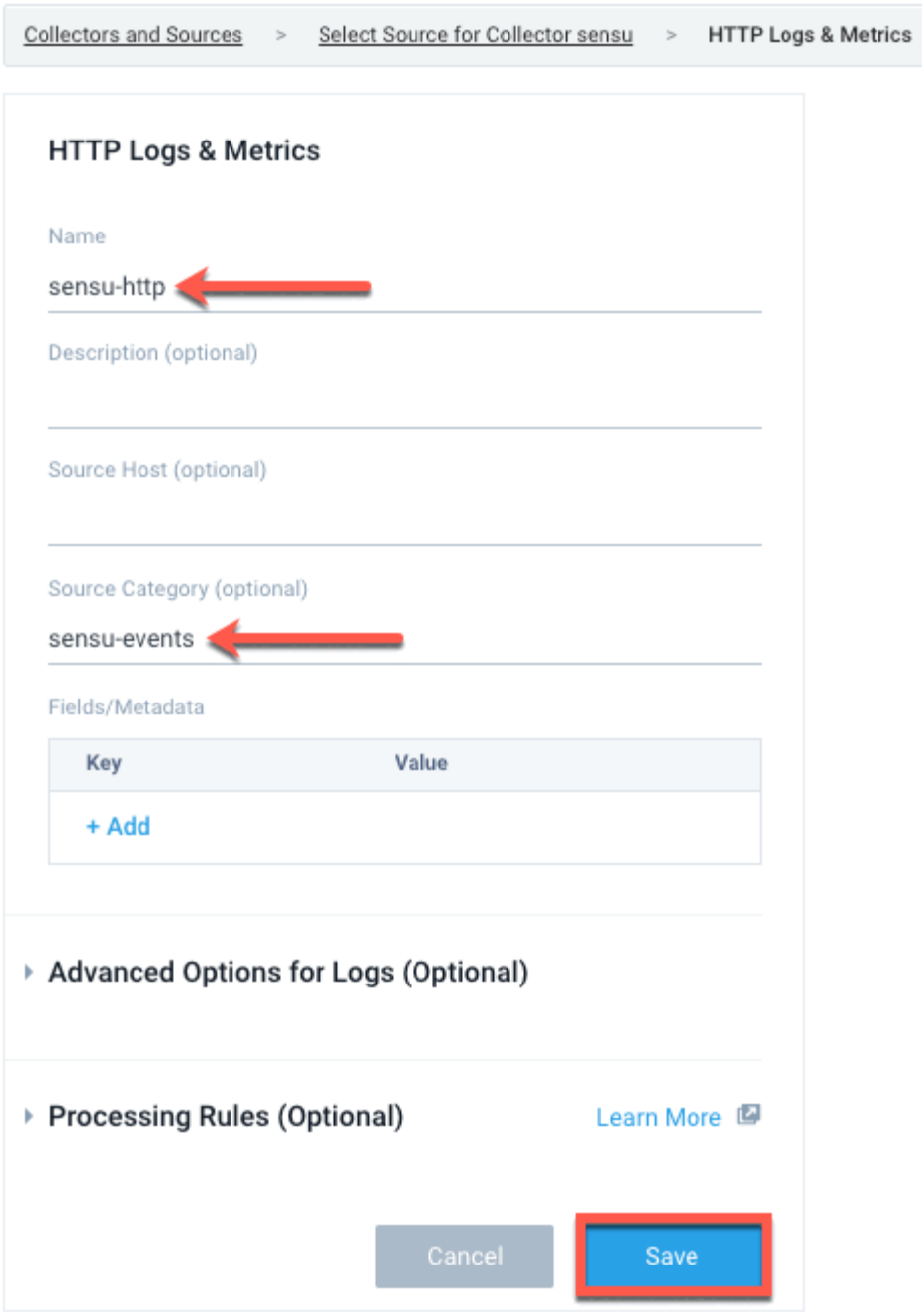


6. Under Cloud APIs, click **HTTP Logs & Metrics**.

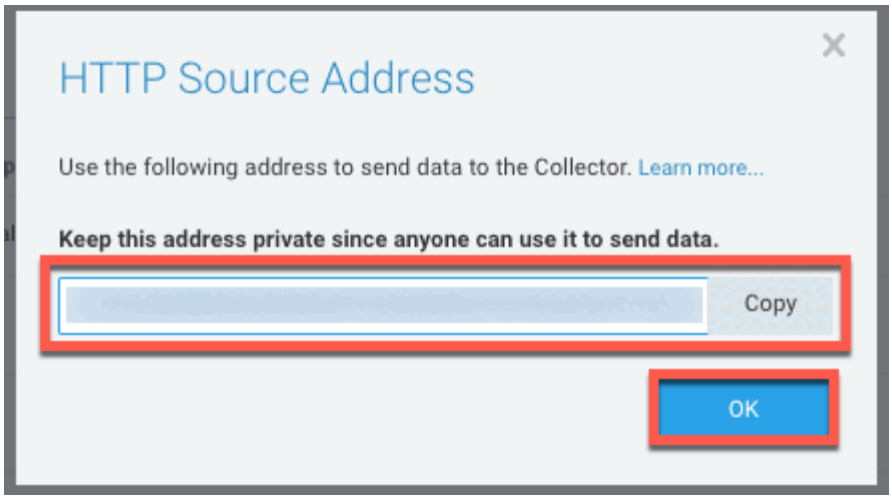


7. In the HTTP Logs & Metrics form:

   - Type **sensu-http** in the Name field.
   - Type **sensu-events** in the Source Category field.
   - Click **Save**.



8. In the HTTP Source Address prompt, copy the listed URL and click OK. You will use this URL in the next step as the `SUMOLOGIC_URL` value for the secret in your Sensu handler definition.



Add the Sumo Logic handler

Now that you've set up a Sumo Logic HTTP Logs and Metrics Source, you can create a handler that uses the sensu/sensu-sumologic-handler dynamic runtime asset to send observability data to Sumo Logic.

The Sensu Sumo Logic Handler asset requires a `SUMOLOGIC_URL` variable. The value for the `SUMOLOGIC_URL` variable is the Sumo Logic HTTP Source Address URL, which you retrieved in the last step of setting up an HTTP Logs and Metrics Source.

> ***NOTE**: This example shows how to set your Sumo Logic HTTP Source Address URL as an environment variable and use it as a secret with Sensu's `Env` secrets provider. Read Use secrets management in Sensu for more information about using the `Env` secrets provider.*

Configure the SUMOLOGIC_URL environment variable

To save your Sumo Logic HTTP Source Address URL as an environment variable:

1. Create the files from which the `sensu-backend` service will read environment variables. If you have already created this file on your system, skip to step 2.

   SHELL

   ```
   sudo touch /etc/default/sensu-backend
   ```

   SHELL

```
sudo touch /etc/sysconfig/sensu-backend
```

2. In the following code, replace `<SumoLogic_HTTPSourceAddress_URL>` with your Sumo Logic HTTP Source Address URL.Run:

   SHELL

   ```
   echo 'SUMOLOGIC_URL=<SumoLogic_HTTPSourceAddress_URL>' | sudo tee -a /etc/default/sensu-backend
   ```

   SHELL

   ```
   echo 'SUMOLOGIC_URL=<SumoLogic_HTTPSourceAddress_URL>' | sudo tee -a /etc/sysconfig/sensu-backend
   ```

3. Restart the sensu-backend:

   ```
   sudo systemctl restart sensu-backend
   ```

This configures the `SUMOLOGIC_URL` environment variable to your Sumo Logic HTTP Source Address URL in the context of the sensu-backend process.

Create the Env secret

Create a secret named `sumologic_url` that refers to the environment variable ID `SUMOLOGIC_URL`. Run:

SHELL

```
cat << EOF | sensuctl create
---
type: Secret
api_version: secrets/v1
metadata:
  name: sumologic_url
spec:
```

```
  id: SUMOLOGIC_URL
  provider: env
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sumologic_url"
  },
  "spec": {
    "id": "SUMOLOGIC_URL",
    "provider": "env"
  }
}
EOF
```

Now you can refer to the `sumologic_url` secret in your handler to securely pass your Sumo Logic HTTP Source Address URL.

Create a Sumo Logic handler

Run the following command to create a handler to send Sensu observability data to your Sumo Logic HTTP Logs and Metrics Source:

SHELL

```
cat << EOF | sensuctl create
---
type: Handler
api_version: core/v2
metadata:
  name: sumologic
spec:
  command: >-
    sensu-sumologic-handler --send-log --send-metrics
```

```
    --source-host "{{ .Entity.Name }}"
    --source-name "{{ .Check.Name }}"
  type: pipe
  runtime_assets:
  - sumologic-handler
  secrets:
  - name: SUMOLOGIC_URL
    secret: sumologic_url
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "sumologic"
  },
  "spec": {
    "command": "sensu-sumologic-handler --send-log --send-metrics --source-host \"{{
.Entity.Name }}\" --source-name \"{{ .Check.Name }}\"",
    "type": "pipe",
    "runtime_assets": [
      "sumologic-handler"
    ],
    "secrets": [
      {
        "name": "SUMOLOGIC_URL",
        "secret": "sumologic_url"
      }
    ]
  }
}
EOF
```

Make sure that your handler was added by retrieving the complete handler definition in YAML or JSON format:

SHELL

```
sensuctl handler info sumologic --format yaml
```

SHELL

```
sensuctl handler info sumologic --format wrapped-json
```

The response will list the complete handler resource definition:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: sumologic
spec:
  command: sensu-sumologic-handler --send-log --send-metrics --source-host "{{
.Entity.Name }}" --source-name "{{ .Check.Name }}"
  env_vars: null
  filters: null
  handlers: null
  runtime_assets:
  - sumologic-handler
  secrets:
  - name: SUMOLOGIC_URL
    secret: sumologic_url
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "sumologic"
  },
  "spec": {
    "command": "sensu-sumologic-handler --send-log --send-metrics --source-host \"{{
```

```
.Entity.Name }}\" --source-name \"{{ .Check.Name }}\"",
    "env_vars": null,
    "filters": null,
    "handlers": null,
    "runtime_assets": [
      "sumologic-handler"
    ],
    "secrets": [
      {
        "name": "SUMOLOGIC_URL",
        "secret": "sumologic_url"
      }
    ],
    "timeout": 0,
    "type": "pipe"
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Create a pipeline with the Sumo Logic handler

With your Sumo Logic handler configured, you can add it to a pipeline workflow. A single pipeline workflow can include one or more event filters, one mutator, and one handler.

To send data for all events (as opposed to only incidents), create a pipeline that includes only the Sumo Logic handler you've already configured and the built-in not_silenced event filter — no mutators. To add the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: sensu_to_sumo
spec:
```

```
  workflows:
  - name: logs_to_sumologic
    filters:
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    handler:
      name: sumologic
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu_to_sumo"
  },
  "spec": {
    "workflows": [
      {
        "name": "logs_to_sumologic",
        "filters": [
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "sumologic",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
EOF
```

Assign the pipeline to a check

To use the `sensu_to_sumo` pipeline, list it in a check definition's pipelines array. This example uses the `check_cpu` check created in Monitor server resources, but you can add the pipeline to any Sensu check you wish. All the observability events that the check produces will be processed according to the pipeline's workflows.

Assign your `sensu_to_sumo` pipeline to the `check_cpu` check to start sending Sensu data to Sumo Logic.

To open the check definition in your text editor, run:

```
sensuctl edit check check_cpu
```

Replace the `pipelines: []` line with the following array:

```
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: sensu_to_sumo
```

To confirm that the updated `check_cpu` resource definition includes the pipelines reference, run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The updated check definition will be similar to this example:

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: check_cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 15
  low_flap_threshold: 0
  output_metric_format: prometheus_text
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: sensu_to_sumo
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

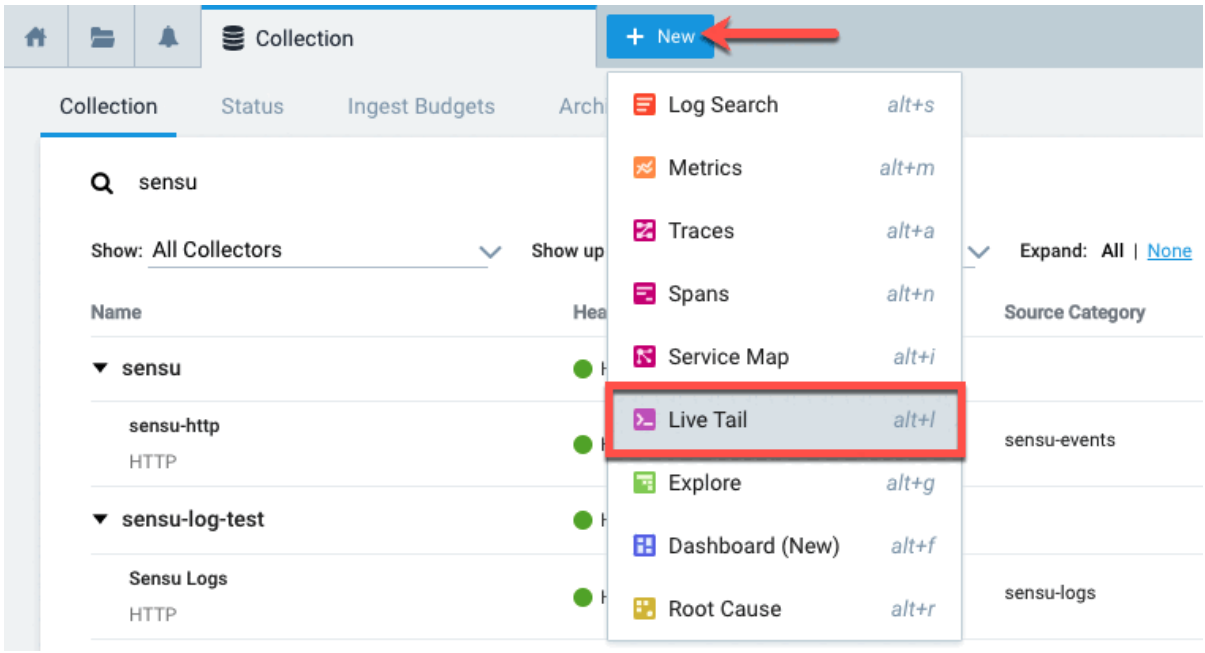
```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "check_cpu",
    "created_by": "admin"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "output_metric_format": "prometheus_text",
    "output_metric_handlers": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "name": "sensu_to_sumo",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

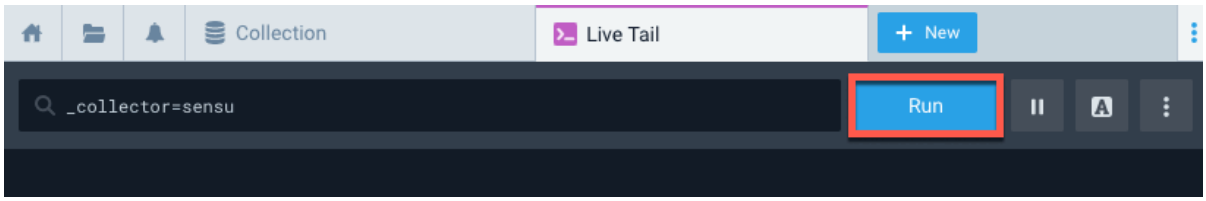
View your Sensu data in Sumo Logic

It will take a few moments after you add the pipeline to the check for your Sensu observability data to appear in Sumo Logic. Use the Live Tail feature to confirm that your data is reaching Sumo Logic.

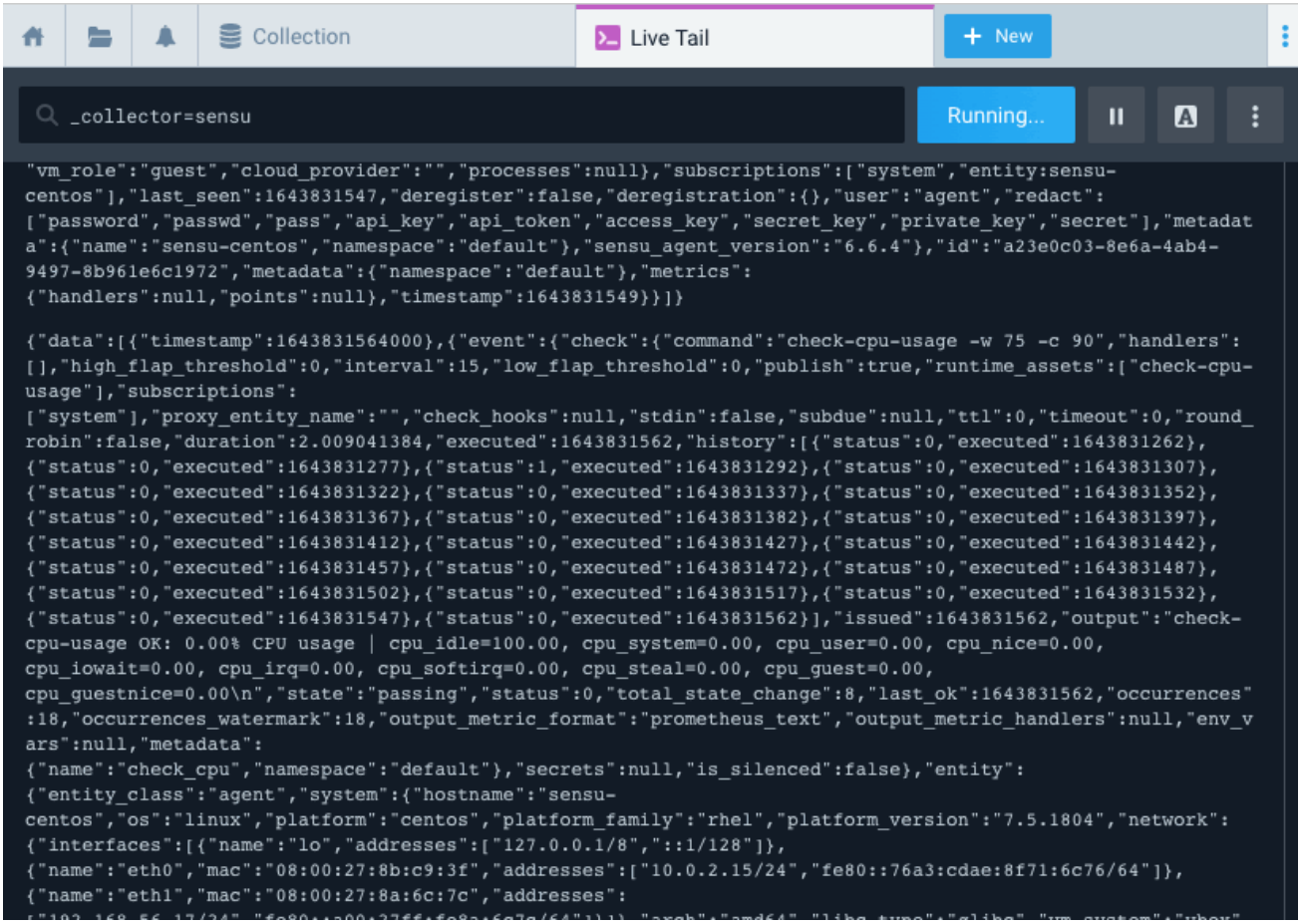
1. In Sumo Logic, click the **+ New** button and select **Live Tail** from the drop-down menu.



2. In the Live Tail search field, enter `_collector=sensu` and click **Run**.



Within a few seconds, the Live Tail page should begin to display your Sensu observability data.



If you see Sensu data on the Live Tail page, well done! You have a successful workflow that sends Sensu observability data to your Sumo Logic account.

Next steps

To share and reuse the check, handler, and pipeline like code, save them to files and start building a monitoring as code repository.

Learn more about the sensu/sensu-sumologic-handler dynamic runtime asset. You can also configure a Sumo Logic dashboard to search, view, and analyze the Sensu data you're sending to your Sumo Logic HTTP Logs and Metrics Source.

In addition to the traditional handler we used in this example, you can use Sensu Plus, our built-in integration, to send metrics to Sumo Logic with a streaming Sumo Logic metrics handler.

Send email alerts with a pipeline

> ***PRO TIP**: You can use the Email Alerts integration in the Sensu Catalog to send email alerts based on Sensu event data instead of following this guide. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Pipelines are Sensu resources composed of observation event processing workflows that include filters, mutators, and handlers. You can use pipelines to send email alerts, create or resolve incidents (in PagerDuty, for example), or store metrics in a time-series database like InfluxDB.

When you are using Sensu in production, events will come from a check or metric you configure. For this guide, you will create an ad hoc event that you can trigger manually to test your email handler.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running on Linux, and install and configure sensuctl.

Your backend will execute a pipeline with a handler that sends notifications to the email address you specify. The pipeline will also include an event filter to make sure you only receive a notification when your event represents a status change.

Add the email handler dynamic runtime asset

Dynamic runtime assets are shareable, reusable packages that help you deploy Sensu plugins. In this guide, you'll use the sensu/sensu-email-handler dynamic runtime asset to power an `email` handler.

Use the following sensuctl example to register the sensu/sensu-email-handler dynamic runtime asset:

```
sensuctl asset add sensu/sensu-email-handler:1.2.2 -r email-handler
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-email-handler:1.2.2
added asset: sensu/sensu-email-handler:1.2.2
```

```
You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["email-handler"].
```

The -r (rename) flag allows you to specify a shorter name for the dynamic runtime asset (in this case, `email-handler`).

You can also download the latest dynamic runtime asset definition for your platform from Bonsai and register the asset with `sensuctl create --file filename.yml`.

To confirm that the handler dynamic runtime asset was added correctly, run:

```
sensuctl asset list
```

The list should include the `email-handler` dynamic runtime asset. For a detailed list of everything related to the asset that Sensu added automatically, run:

```
sensuctl asset info email-handler
```

The sensu/sensu-email-handler dynamic runtime asset includes the `sensu-email-handler` command, which you will use when you create the email handler definition later in this guide.

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read the asset reference for more information about dynamic runtime asset builds.*

Add an event filter

Event filters allow you to fine-tune how your events are handled and reduce alert fatigue. In this guide, your event filter will send notifications only when your event's state changes (for example, for any change between `0` OK, `1` warning, and `2` critical).

Here's an overview of how the `state_change_only` filter will work:

- If your event status changes from `0` to `1`, you will receive **one** email notification for the change to warning status.
- If your event status stays at `1` for the next hour, you **will not** receive repeated email notifications during that hour.
- If your event status changes to `2` after 1 hour at `1`, you will receive **one** email notification for the change from warning to critical status.
- If your event status fluctuates between `0`, `1`, and `2` for the next hour, you will receive **one** email notification **each time** the status changes.

To create the event filter, run:

TEXT

```
cat << EOF | sensuctl create
---
type: EventFilter
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: state_change_only
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1
  runtime_assets: []
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "annotations": null,
    "labels": null,
    "name": "state_change_only"
```

```
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": [

    ]
  }
}
EOF
```

Create the email handler definition

After you add an event filter, create the email handler definition to specify the email address where the handler will send notifications. In the handler definition's `command` value, you'll need to change a few things:

- `<sender@example.com>`: Replace with the email address you want to use to send email alerts.
- `<recipient@example.com>`: Replace with the email address you want to receive email alerts.
- `<smtp_server@example.com>`: Replace with the hostname of your SMTP server.
- `<username>`: Replace with your SMTP username, typically your email address.
- `<password>`: Replace with your SMTP password, typically the same as your email password.

***NOTE**: To use Gmail or G Suite as your SMTP server, follow Google's instructions to send email via SMTP. If you have enabled 2-step verification on your Google account, use an app password instead of your login password. If you have not enabled 2-step verification, you may need to adjust your app access settings to follow the example in this guide.*

After you update the command with your email, server, username, and password values in the example below, run the updated code to create the email handler definition:

TEXT

```
cat << EOF | sensuctl create
```

```
---
api_version: core/v2
type: Handler
metadata:
  name: email
spec:
  type: pipe
  command: sensu-email-handler -f <sender@example.com> -t <recipient@example.com> -s
<smtp_server@example.com> -u username -p password
  timeout: 10
  runtime_assets:
  - email-handler
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "api_version": "core/v2",
  "type": "Handler",
  "metadata": {
    "name": "email"
  },
  "spec": {
    "type": "pipe",
    "command": "sensu-email-handler -f <sender@example.com> -t
<recipient@example.com> -s <smtp_server@example.com> -u username -p password",
    "timeout": 10,
    "runtime_assets": [
      "email-handler"
    ]
  }
}
EOF
```

The sensu/sensu-email-handler dynamic runtime asset makes it possible to add a template that provides context for your email notifications. The email template functionality uses tokens to populate the values provided by the event, and you can use HTML to format the email.

Create a pipeline

With your event filter and handler configured, you can build a pipeline workflow. A single pipeline workflow can include one or more filters, one mutator, and one handler.

In this case, the pipeline includes your `state_change_only` event filter and `email` handler, as well as two built-in event filters, is_incident and not_silenced. These two built-in filters are included in every Sensu backend installation, so you don't have to create them. The is_incident and not_silenced event filters ensure that you receive alerts for unsilenced events with *only* warning (`1`) or critical (`2`) status:

To create the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: alerts_pipeline
spec:
  workflows:
  - name: email_alerts
    filters:
    - name: state_change_only
      type: EventFilter
      api_version: core/v2
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    handler:
      name: email
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "alerts_pipeline"
  },
  "spec": {
    "workflows": [
      {
        "name": "email_alerts",
        "filters": [
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
EOF
```

Before your pipeline can send alerts to your email, you need an event that generates the alerts.In the

final step, you will create an ad hoc event that you can trigger manually.

Create and trigger an ad hoc event

To create an ad hoc event, first use `sensuctl env` to set up environment variables. The environment variables will provide the required Sensu API access token credential for the Sensu API:

```
eval $(sensuctl env)
```

Verify that the `SENSU_ACCESS_TOKEN` environment variable is set by echoing its value:

```
echo $SENSU_ACCESS_TOKEN
```

The response will list the `SENSU_ACCESS_TOKEN` value.

With the environment variables set, you can use the Sensu API to create your ad hoc observability event.

> ***NOTE**: The example events use the default namespace. If you are using a different namespace, replace `default` in the event definitions and the API URLs with the name of the desired namespace.*

This event outputs the message “Everything is OK.” when it occurs:

```
curl -sS -H 'Content-Type: application/json' \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server01",
      "namespace": "default"
    }
  },
```

```
  "check": {
    "metadata": {
      "name": "server-health"
    },
    "output": "Everything is OK.",
    "status": 0
  }
}' \
http://localhost:8080/api/core/v2/namespaces/default/events
```

As configured, the event status is `0` (OK). Now it's time to trigger an event and view the results!

To generate a status change event, use the update event endpoint to create a `1` (warning) event. Run:

```
curl -sS -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server01",
      "namespace": "default"
    }
  },
  "check": {
    "metadata": {
      "name": "server-health"
    },
    "output": "This is a warning.",
    "status": 1
  },
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "alerts_pipeline"
    }
  ]
}' \
```

```
http://localhost:8080/api/core/v2/namespaces/default/events/server01/server-health
```

***NOTE**: If you receive an `invalid credentials` error, refresh your token.Run `eval $(sensuctl env)`.*

Check your email — you should receive a message from Sensu!

Create another event with status set to `0`. Run:

```
curl -sS -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server01",
      "namespace": "default"
    }
  },
  "check": {
    "metadata": {
      "name": "server-health"
    },
    "output": "Everything is OK.",
    "status": 0
  },
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "alerts_pipeline"
    }
  ]
}' \
http://localhost:8080/api/core/v2/namespaces/default/events/server01/server-health
```

You should receive another email because the event status changed to `0` (OK).

Next steps

Now that you know how to apply a handler to a check and take action on events:

- Reuse this email handler with the `check_cpu` check from our Monitor server resources guide.
- Learn how to use the event filter, handler, and pipeline resources you created to start developing a monitoring as code repository.
- Read the pipelines reference for in-depth pipeline documentation.
- Check out Route alerts with event filters for a complex pipeline example that includes several workflows with different event filters and handlers.

Send PagerDuty alerts with Sensu

> ***PRO TIP**: You can use the PagerDuty integration in the Sensu Catalog to send alerts based on Sensu event data instead of following this guide. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Follow this guide to create a pipeline that sends incident alerts to PagerDuty. Sensu checks are commands the Sensu agent executes that generate observability data in a status or metric event. Sensu pipelines define the event filters and actions the Sensu backend executes on the events.

This guide will help you send alerts to PagerDuty by configuring a pipeline and adding it to a check named `check_cpu`. If you don't already have this check in place, follow Monitor server resources to add it.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl. You'll also need your PagerDuty API integration key to set up the handler in this guide.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition. To run the `check_cpu` check, you'll need a Sensu entity with the subscription `system`.

First, find your entity name:

```
sensuctl entity list
```

The `ID` in the response is the name of your entity.

Replace `<ENTITY_NAME>` with the name of your entity in the sensuctl command below. Then run the command to add the `system` subscription to your entity:

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `system` and press enter.

Confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register the dynamic runtime asset

The sensu/sensu-pagerduty-handler dynamic runtime asset includes the scripts you will need to send events to PagerDuty.

To add the sensu/sensu-pagerduty-handler asset, run:

```
sensuctl asset add sensu/sensu-pagerduty-handler:2.2.0 -r pagerduty-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `pagerduty-handler`.

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-pagerduty-handler:2.2.0
added asset: sensu/sensu-pagerduty-handler:2.2.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["pagerduty-handler"].
```

To confirm that the asset was added to your Sensu backend, run:

```
sensuctl asset info pagerduty-handler
```

The response will list the available builds for the Sensu PagerDuty Handler dynamic runtime asset.

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read the asset reference for more information about dynamic runtime asset builds.*

Add the PagerDuty handler

Now that you've added the dynamic runtime asset, you can create a handler that uses the asset to send non-OK events to PagerDuty.

In the following command, replace `<PAGERDUTY_KEY>` with your PagerDuty API integration key. Then run the updated command:

```
sensuctl handler create pagerduty \
--type pipe \
--runtime-assets pagerduty-handler \
--command "sensu-pagerduty-handler -t <PAGERDUTY_KEY>"
```

***NOTE**: For checks whose handlers use the Sensu PagerDuty Handler dynamic runtime asset (like the one you've created in this guide), you can use an alternative method for authenticating and routing alerts based on PagerDuty teams.*

To use this option, list the teams' PagerDuty API integration keys in the handler definition as environment variables or secrets or in the `/etc/default/sensu-backend` configuration file as environment variables. Then, add check or agent annotations to specify which PagerDuty teams should receive alerts based on check events. Sensu will look up the key in the handler definition or backend configuration file that corresponds to the team name in the check annotation to authenticate and send alerts.

Make sure that your handler was added by retrieving the complete handler definition in YAML or JSON format:

SHELL

```
sensuctl handler info pagerduty --format yaml
```

SHELL

```
sensuctl handler info pagerduty --format wrapped-json
```

The response will list the complete handler resource definition:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: pagerduty
spec:
  command: sensu-pagerduty-handler -t <PAGERDUTY_KEY>
  env_vars: null
  handlers: null
  runtime_assets:
  - pagerduty-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "pagerduty"
  },
  "spec": {
    "command": "sensu-pagerduty-handler -t <PAGERDUTY_KEY>",
```

```
    "env_vars": null,
    "handlers": null,
    "runtime_assets": [
      "pagerduty-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

Create a pipeline with event filters and a handler

With your handler configured, you can add it to a pipeline workflow. A single pipeline workflow can include one or more filters, one mutator, and one handler.

In this case, the pipeline includes the built-in is_incident and not_silenced event filters, as well as the `pagerduty` handler you've already configured. To create the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: cpu_check_alerts
spec:
  workflows:
  - name: pagerduty_alerts
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    - name: not_silenced
      type: EventFilter
```

```
      api_version: core/v2
    handler:
      name: pagerduty
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "cpu_check_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "pagerduty_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "pagerduty",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
EOF
```

Assign the pipeline to a check

To use the `cpu_check_alerts` pipeline, list it in a check definition's pipelines array (in this case, the `check_cpu` check created in Monitor server resources).All the observability events that the check produces will be processed according to the pipeline's workflows.

Assign your `cpu_check_alerts` pipeline to the `check_cpu` check to receive Slack alerts when the CPU usage of your system reaches the specific thresholds set in the check command.

To open the check definition in your text editor, run:

```
sensuctl edit check check_cpu
```

Replace the `pipelines: []` line with the following array:

```
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: cpu_check_alerts
```

To view the updated `check_cpu` resource definition, run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The updated check definition will be similar to this example:

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: cpu_check_alerts
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu"
  },
  "spec": {
```

```
      "check_hooks": null,
      "command": "check-cpu-usage -w 75 -c 90",
      "env_vars": null,
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 10,
      "low_flap_threshold": 0,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "pipelines": [
        {
          "api_version": "core/v2",
          "name": "cpu_check_alerts",
          "type": "Pipeline"
        }
      ],
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "secrets": null,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "ttl": 0
    }
  }
```


Observe an alert in PagerDuty

It might take a few moments after you add the pipeline to the check for the check to be scheduled on entities with the `system` subscription and the result sent back to Sensu backend.

As configured, the `cpu_check` command requires CPU usage to reach 75% capacity to send a non-

OK event. To trigger an alert and confirm that the check and pipeline are working properly, adjust the check command to reduce the usage percentage required for a non-OK event.

To open the check definition in your text editor, run:

```
sensuctl edit check check_cpu
```

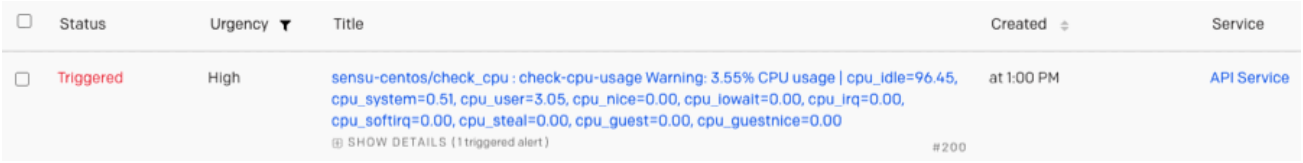
Replace the `-w` value in the `command` line with `1` and save the updated check definition:

```
  command: check-cpu-usage -w 1 -c 90
```

You should see a response to confirm the update:

```
Updated /api/core/v2/namespaces/default/checks/check_cpu
```

After Sensu detects a non-OK event, the handler in your pipeline will send the alert to your PagerDuty account, where you should see an event similar to this one:



Resolve the alert in PagerDuty

To complete your workflow, restore the CPU usage command to 75% so Sensu sends a resolution to PagerDuty.Open the check definition in your text editor:

```
sensuctl edit check check_cpu
```

Replace the `-w` value in the `command` line with `75` and save the updated check definition:

```
```

```
    command: check-cpu-usage -w 75 -c 90
```

In your PagerDuty account, the alert should now be listed under the “Resolved” tab.

To view the resolved event with sensuctl, run:

```
sensuctl event list
```

The response should show that `cpu_check` has an OK (0) status:

```
    Entity       Check                                          Output
Status   Silenced          Timestamp                       UUID
─────────────── ─────────── 
─────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────────────────────────────────
────────────────────────────── ─────── ───────── ──────────────────────────────
────────────────────────────────────
  sensu-centos   check_cpu   check-cpu-usage OK: 4.17% CPU usage | cpu_idle=95.83, cpu_system=1.04, cpu_user=3.13,
cpu_nice=0.00, cpu_iowait=0.00, cpu_irq=0.00, cpu_softirq=0.00, cpu_steal=0.00, cpu_guest=0.00, cpu_guestnice=0.00
0   false      2021-11-17 21:09:07 +0000 UTC   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The web UI Events page will also show that the `cpu_check` check is passing.

Next steps

You should now have a working set-up with a check and a pipeline that sends alerts to your PagerDuty account. To share and reuse the check, handler, and pipeline like code, save them to files and start building a monitoring as code repository.

You can customize your PagerDuty handler with configuration options like severity mapping, PagerDuty team-based routing and authentication via check and agent annotations, and event-based templating. Learn more about the Sensu PagerDuty integration and our curated, configurable quick-start template for incident management to integrate Sensu with your existing PagerDuty workflows.

Send Slack alerts with a pipeline

> ***PRO TIP**: You can use the Slack Alerts integration in the Sensu Catalog to send Slack alerts based on Sensu event data instead of following this guide. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Pipelines are Sensu resources composed of observation event processing workflows that include filters, mutators, and handlers. You can use pipelines to send email alerts, create or resolve incidents (in PagerDuty, for example), or store metrics in a time-series database like InfluxDB.

This guide will help you send alerts to Slack in the channel `monitoring` by configuring a pipeline and adding it to a check named `check_cpu`. If you don't already have this check in place, follow Monitor server resources to add the check.

Before you start, follow the RHEL family install instructions to install and configure the Sensu backend, the Sensu agent, and sensuctl.

Configure a Sensu entity

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition. To run the `check_cpu` check, you'll need a Sensu entity with the subscription `system`.

First, find your entity name:

```
sensuctl entity list
```

The `ID` in the response is the name of your entity.

Replace `<ENTITY_NAME>` with the name of your entity in the sensuctl command below. Then run the command to add the `system` subscription to your entity:

```
```

```
sensuctl entity update <ENTITY_NAME>
```

- For `Entity Class`, press enter.
- For `Subscriptions`, type `system` and press enter.

Confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

The response should indicate `active (running)` for both the Sensu backend and agent.

Register the dynamic runtime asset

Dynamic runtime assets are shareable, reusable packages that help you deploy Sensu plugins. In this guide, you'll use the sensu/sensu-slack-handler dynamic runtime asset to power a `slack` handler.

Use `sensuctl asset add` to register the sensu/sensu-slack-handler dynamic runtime asset:

```
sensuctl asset add sensu/sensu-slack-handler:1.5.0 -r sensu-slack-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `sensu-slack-handler`.

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-slack-handler:1.5.0
added asset: sensu/sensu-slack-handler:1.5.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-slack-handler"].
```

You can also download the latest dynamic runtime asset definition for your platform from Bonsai and register the asset with `sensuctl create --file filename.yml` or `sensuctl create --file filename.json`.

> ***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Get a Slack webhook

If you're already the admin of a Slack, visit `https://YOUR_WORKSPACE_NAME_HERE.slack.com/services/new/incoming-webhook` and follow the steps to add the Incoming WebHooks integration, choose a channel, and save the settings.If you're not yet a Slack admin, create a new workspace and then create and save your webhook.

After you save your webhook, you can find the webhook URL under **Integration Settings**.

Create a handler

Use sensuctl to create a handler called `slack` that pipes observation data (events) to Slack using the sensu/sensu-slack-handler dynamic runtime asset.Before you run the sensuctl command below, edit it to include your Slack webhook URL and the channel where you want to receive events:

```
sensuctl handler create slack \
--type pipe \
--env-vars "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX"
\
--command "sensu-slack-handler --channel '#monitoring'" \
--runtime-assets sensu-slack-handler
```

You should receive a confirmation message:

```
Created
```

The `sensuctl handler create slack` command creates a handler resource. To view the `slack` handler definition, run:

SHELL

```
sensuctl handler info slack --format yaml
```

SHELL

```
sensuctl handler info slack --format wrapped-json
```

The `slack` handler resource definition will be similar to this example:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: slack
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters: null
  handlers: null
  runtime_assets:
  - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack"
  },
```

```
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX"
    ],
    "filters": null,
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

***PRO TIP**: You can also view complete resource definitions in the Sensu web UI.*

You can share and reuse this handler like code — save it to a file and start building a monitoring as code repository.

Create a pipeline that includes the handler

With your handler configured, you can add it to a pipeline workflow. A single pipeline workflow can include one or more filters, one mutator, and one handler.

For now, the pipeline includes only the `slack` handler and the built-in not_silenced event filter so that you receive an alert for every event the check generates (including events with OK status). To create the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: cpu_check_alerts
```

```
spec:
  workflows:
  - name: slack_alerts
    filters:
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "cpu_check_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "slack_alerts",
        "filters": [
          {
            "name": "not_silenced",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
```

```
EOF
```

Assign the pipeline to a check

To use the `cpu_check_alerts` pipeline, list it in a check definition's pipelines array (in this case, the `check_cpu` check created in Monitor server resources).All the observability events that the check produces will be processed according to the pipeline's workflows.

Assign your `cpu_check_alerts` pipeline to the `check_cpu` check to receive Slack alerts when the CPU usage of your system reaches the specific thresholds set in the check command.

To open the check definition in your text editor, run:

```
sensuctl edit check check_cpu
```

Replace the `pipelines: []` line with the following array and save the updated check definition:

SHELL

```
  pipelines:
  - type: Pipeline
    api_version: core/v2
    name: cpu_check_alerts
```

SHELL

```
  "pipelines": [
    {
      "type": "Pipeline",
      "api_version": "core/v2",
      "name": "cpu_check_alerts"
    }
  ]
```

You should see a response to confirm the update:

```
Updated /api/core/v2/namespaces/default/checks/check_cpu
```

To view the updated `check_cpu` resource definition, run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The updated check definition will be similar to this example:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: cpu_check_alerts
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
```

```
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

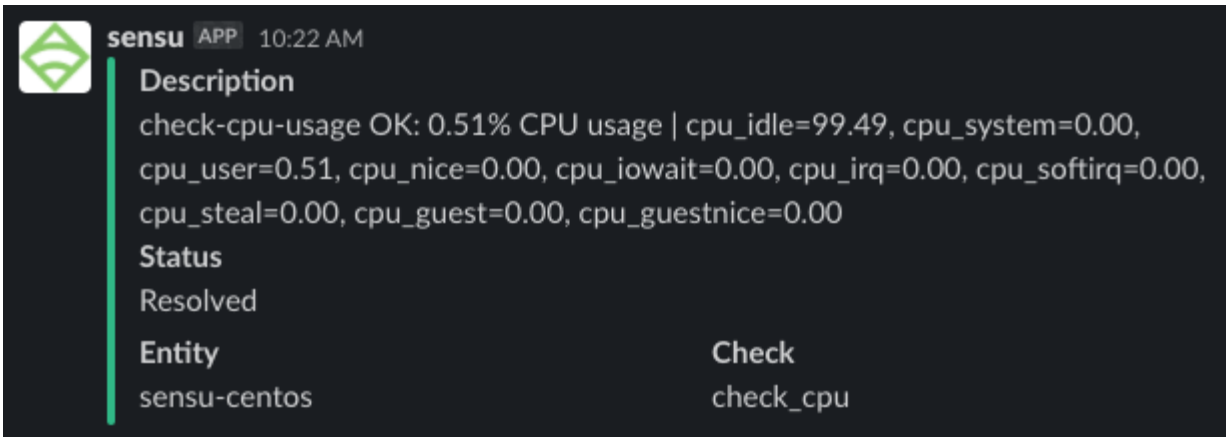
JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "name": "cpu_check_alerts",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
```

```
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "ttl": 0
    }
  }
```

Validate the pipeline

It might take a few moments after you add the pipeline to the check for the check to be scheduled on entities with the `system` subscription and the result sent back to Sensu backend. After an event is handled, you should receive a message like this in Slack:



Verify proper handler behavior with `sensu-backend` logs. Read Troubleshoot Sensu for log locations by platform.

Whenever an event is being handled, a log entry is added with the message `"handler":"slack","level":"debug","msg":"sending event to handler"`, followed by a second log entry with the message `"msg":"event pipe handler executed","output":"","status":0`.

Add another event filter to the pipeline

At this point, the `cpu_check_alerts` pipeline has probably sent quite a few Slack messages for events with OK (`0`) status. To receive alerts for events with *only* warning (`1`) or critical (`2`) status,

add the built-in is_incident event filter to the pipeline:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
metadata:
  name: cpu_check_alerts
spec:
  workflows:
  - name: slack_alerts
    filters:
    - name: not_silenced
      type: EventFilter
      api_version: core/v2
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: slack
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "cpu_check_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "slack_alerts",
        "filters": [
          {
            "name": "not_silenced",
```

```
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
}
EOF
```

Adding the is_incident filter to your pipeline should quickly reduce the number of alerts you receive in Slack.

Next steps

Now that you know how to apply a pipeline to a check and take action on events, read the pipelines reference for in-depth documentation.Read Route alerts with event filters for a more complex example with multiple filters and handlers organized into several pipeline workflows.

For more information about customizing your Slack alerts, read the Sensu Slack Handler page in Bonsai.

Follow Send PagerDuty alerts with Sensu to configure a check that generates status events and a handler that sends Sensu alerts to PagerDuty for non-OK events.

Operations

The Operations category will help you get Sensu up and running, from your first installation in your local development environment through a large-scale Sensu deployment using secrets management. You'll also learn how to keep your Sensu implementation running, with guides for upgrading, monitoring, and troubleshooting.

Monitoring as code

Monitoring as code with Sensu explains how to use Sensu's end-to-end monitoring as code approach to manage your observability configuration the same way you build, test, and deploy your applications and infrastructure.

Deploy Sensu

Deploy Sensu describes how to plan, install, configure, and deploy Sensu's flexible monitoring and observability pipeline.

To plan your Sensu deployment, read the hardware requirements and deployment architecture pages. To start using Sensu locally or in development environments, follow the steps in the Install Sensu guide.

When you're ready to deploy Sensu in production, learn to generate certificates, secure your Sensu installation, run a Sensu cluster, and reach multi-cluster visibility. You'll also find guides for scaling your implementation with Sensu's Enterprise datastore and using configuration management tools to ensure repeatable Sensu deployments and consistent configuration.

Control Access

Control Access explains how Sensu administrators control access by authentication (verifying user identities) and authorization (establishing and managing user permissions for Sensu resources).

Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool. Use Sensu's built-in basic authentication or configure external authentication

providers via Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) to authenticate your Sensu users.

Next, learn to configure authorization for your authenticated Sensu users with role-based access control (RBAC) and set up user permissions for interacting with Sensu resources.

Maintain Sensu

Maintain Sensu includes upgrade, migration, troubleshooting, and license management information to keep your Sensu implementation running smoothly.

Follow our step-by-step instructions to upgrade to the latest version of Sensu from any earlier version. If you're still using Sensu Core or Sensu Enterprise, read Migrate from Sensu Core and Sensu Enterprise to Sensu Go. You can also learn how to activate and and view your commercial Sensu license or troubleshoot to identify and resolve problems with your Sensu implementation, from reading and configuring Sensu service logs to using Sensu handlers and filters to test and debug your implementation.

Monitor Sensu

Monitor Sensu covers how to log Sensu services, monitor your Sensu backend with a secondary instance, and retrieve and process health data for your Sensu cluster. You can also learn about Tessen the Sensu call-home service, which helps us understand how Sensu is being used and make informed decisions about product improvements.

Manage Secrets

Manage Secrets explains how to use Sensu's secrets management to eliminate the need to expose secrets like usernames, passwords, and access keys in your Sensu configuration. Learn to configure secrets and secrets providers resources to obtain secrets from one or more external secrets providers, refer to external secrets, and consume secrets via backend environment variables.

Monitoring as code with Sensu

Sensu's end-to-end monitoring as code solution allows you to manage your monitoring and observability configurations the same way you build, test, and deploy your applications and infrastructure, like Kubernetes and Terraform. Monitoring as code combines composable building blocks with robust APIs so you can define your entire observability configuration as declarative YAML or JSON code and improve visibility, reliability, portability, and repeatability.

When a new endpoint starts up, like a cloud compute instance or Kubernetes Pod, the endpoint's agent automatically registers it with the platform and starts collecting observability data according to the code in your configuration files. Teams can share and remix observability configurations for collecting events and metrics, diagnosing issues, sending alerts, and automatically remediating problems.

- Share, edit, review, and version your observability configuration files just like you would with other "as code" solutions, within one team or among teams across your organization.
- Maintain revision control and change history for your observability configurations.
- Export the Sensu configuration for one environment and replicate the same configuration in other environments.
- Remove, restore, back up, and recover Sensu instances based on your Sensu configuration files.
- Include your observability configuration in your centralized continuous integration/continuous delivery (CI/CD) pipeline to keep your configuration files aligned with your product and services.

To get started with monitoring as code, you'll need a repository and configuration files that contain your resource definitions.

Create a monitoring as code repository

Create a monitoring as code repository for the configuration files that contain the Sensu resource definitions you use for monitoring and observability. You can use any source control repository.

The way you will use your configuration files will help you choose the best structure for your monitoring as code repository. For example, if you are likely to share observability components or manage your

configuration files as part of your CI/CD workflow, it probably makes sense to use individual configuration files for different types of resources: one file for all of your checks, one file for all of your handlers, and so on. If you want to facilitate more granular sharing, you can save one resource definition per file.

If you want to share complete end-to-end observability configurations with your colleagues, you might save all of the resource definitions for each observability configuration in a single configuration file. This allows others to read through an entire configuration without interruption, and it's convenient for demonstrating a complete Sensu configuration. However, a single configuration file that includes every resource type isn't the best structure for CI/CD management or sharing resources among teams.

SensuFlow, our GitHub Action for managing Sensu resources via repository commits, requires a repository structure organized by clusters and namespaces. All resources of each type for each namespace are saved in a single configuration file:

```
.sensu/
  cluster/
    namespaces.yml
  namespaces/
    <namespace>/
      checks/
      hooks/
      filters/
      handlers/
      handlersets/
      mutators/
      pipelines/
```

Adopt a configuration file strategy

Configuration files contain your Sensu resource definitions. You can build configuration files as you go, adding resource definitions as you create them. You can also create your entire observability configuration first, then export some or all of your resource definitions to a file. Or, you can use a mix: export all of your existing resource definitions to configuration files and append new resources as you create them.

When you are ready to replicate your exported resource definitions, use `sensuctl create`.

***NOTE**: You cannot replicate API key or user resources from a `sensuctl dump` export.*

API keys must be reissued, but you can use your exported configuration file as a reference for granting new API keys to replace the exported keys.

When you export users, required password attributes are not included. You must add a `password_hash` or `password` to `users` resources before replicating them with the `sensuctl create` command.

Build as you go

To build as you go, use sensuctl commands to retrieve your Sensu resource definitions as you create them and copy the definitions into your configuration files.

For example, if you follow Monitor server resources and create a check named `check_cpu`, you can retrieve that check definition in YAML or JSON format with sensuctl:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The sensuctl response will include the complete `check_cpu` resource definition in the specified format:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
```

```
  handlers: null
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines:
  - api_version: core/v2
    name: reduce_alerts
    type: Pipeline
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": null,
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [
```

```
      {
        "api_version": "core/v2",
        "name": "reduce_alerts",
        "type": "Pipeline"
      }
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

If you prefer, you can also view JSON resource definitions in the Sensu web UI.

You can copy these resource definitions and paste them into manually created configuration files located anywhere on your system.

Alternatively, you can view resource definitions and copy them into a new or existing configuration file with a single sensuctl command. To use the following examples, replace `<resource>` with the resource type (like `check`) and replace `<resource_name>` with the name of the resource (like `check_cpu`).

- Copy the resource defintion to a new file (or overwrite an existing file with the same name):

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format yaml > resource.yml
  ```

  SHELL

```
sensuctl <resource> info <resource_name> --format wrapped-json >
resource.json
```

- Copy the resource defintion to a new file (or overwrite an existing file with the same name) and show the resource definition in stdout:

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format yaml | tee resource.yml
  ```

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format wrapped-json | tee
  resource.json
  ```

- Append the resource defintion to an existing file:

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format yaml >> resource.yml
  ```

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format wrapped-json >>
  resource.json
  ```

- Append the resource defintion to an existing file and show the resource definition in stdout:

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format yaml | tee -a resource.yml
  ```

  SHELL

  ```
  sensuctl <resource> info <resource_name> --format wrapped-json | tee -a
  resource.json
  ```

Export existing resources

If you've already created observability resources, use `sensuctl dump` to create a copy of your existing resource definitions.

First, create a sensu directory:

```
mkdir sensu
```

Then, copy your observability resource definitions according to the repository structure you are using. For example, if you want to save resources according to type and namespace, this command will save all of your check definitions for the `production` namespace in one configuration file:

SHELL

```
sensuctl dump core/v2.CheckConfig \
--namespace production \
--format yaml | > sensu/namespaces/production/checks.yml
```

SHELL

```
sensuctl dump core/v2.CheckConfig \
--namespace production \
--format wrapped-json | > sensu/namespaces/production/checks.json
```

Repeat this command for each resource type in each of your namespaces.

Strip namespaces from resource definitions

To replicate and reuse resources in any namespace without manual editing, create a copy of your existing resources with the namespaces stripped from their definitions:

SHELL

```
sensuctl dump all \
```

```
--all-namespaces \
--format yaml | grep -v "^\s*namespace:" > sensu/resources.yml
```

SHELL

```
sensuctl dump all \
--all-namespaces \
--format wrapped-json | grep -v "^\s*namespace:" > sensu/resources.json
```

Best practices for monitoring as code

Sensu's monitoring as code solution is flexible — you can use any source control repository and choose your own directory structure — but following a few best practices will contribute to a successful implementation.

- To maintain consistency, save all of your resources as only one file type: YAML or JSON.
- Include all dependencies within a resource definition. For example, if a handler requires a dynamic runtime asset and a secret, include the asset and secret definitions with the definition for the handler itself.
- Choose the labels you use in your resource definitions with care. CI/CD systems like SensuFlow rely on labels to determine which resources to delete, so if all of your resources have the same labels, you could delete resources you didn't intend to be managed in a particular CI/CD workflow.
- Establish a resource-labeling schema throughout your organization to facilitate CI/CD. Following the same method for applying labels helps keep unmanaged Sensu resources from multiplying and allows different teams to confidently deploy their own CI/CD workflows without the risk of accidentally deleting another team's resources.

Implement CI/CD with monitoring as code

When you're ready, expand your monitoring as code practices to include managing your Sensu configuration files with a CI/CD workflow. CI/CD takes the manual work out of maintaining and updating your monitoring as code repository so that any updates to the Sensu resources in your monitoring as code repository are reflected in your Sensu configuration in a timely manner.

If you're already using CI/CD, you already have workflows for versioning, building, testing, and

deploying your code. Integrating monitoring as code means your monitoring and observability can go through those same CI/CD workflows.

There's no one "correct" way to implement CI/CD with monitoring as code, but the SensuFlow GitHub Action offers a turnkey reference implementation that helps you create your own monitoring as code workflow and start managing Sensu resources via repository commits.

Use SensuFlow for CI/CD monitoring as code

SensuFlow is a git-based, prescriptive monitoring as code workflow that uses sensuctl (including sensuctl prune) to synchronize your monitoring and observability code with your Sensu deployments.

> ***NOTE**: SensuFlow is available for technical preview, and individual components in the workflow may change. Before you use SensuFlow in production, test it in a development environment or a dedicated test namespace in your current environment.*

SensuFlow requires:

- A code repository of Sensu resource definitions
- A Sensu role-based access control (RBAC) service account with permission to manage all resources in your repository
- A resource labeling convention to designate which resources the SensuFlow workflow should manage
- Integration with your CI/CD system to run sensuctl commands as the service account user from the repository of resource definitions

Read the SensuFlow GitHub Action marketplace page and Monitoring as code with Sensu Go and SensuFlow to get started with SensuFlow as your monitoring as code workflow.

Deploy Sensu

Use the information and instructions in the Deploy Sensu category to plan, install, configure, and deploy Sensu's flexible monitoring and observability pipeline.

Plan your Sensu deployment

Find Sensu agent and backend requirements and networking and cloud recommendations in the hardware requirements.

Deployment architecture for Sensu describes planning considerations and recommendations for a production-ready Sensu deployment, along with communication security details and diagrams showing single, clustered, and large-scale deployment architectures.

Install Sensu

When you're ready to start using Sensu, the pathway you follow will depend on your monitoring and observability needs. No matter which pathway you choose, you should begin with the Install Sensu guide. If you just want to use Sensu locally, you can do that by installing Sensu according to the steps in the guide. You can also use the Install Sensu guide to set up proof-of-concept and testing in a development environment.

Deploy Sensu in production

To deploy Sensu for use outside of a local development environment, install Sensu and follow these guides to achieve a production-ready installation:

1. Generate certificates, which you will need to secure a Sensu cluster and its agents.
2. Secure your Sensu installation using the certificates you generate to make Sensu production-ready.
3. Run a Sensu cluster, a group of three or more sensu-backend nodes connected to a shared database, to improve Sensu's availability, reliability, and durability.
4. Reach multi-cluster visibility with federation so you can gain visibility into the health of your infrastructure and services across multiple distinct Sensu instances within a single web UI and

mirror your changes in one cluster to follower clusters.

Read the etcd replicators reference to learn how the etcd-replicators datatype in the enterprise/federation/v1 API allows you to manage role-based access control (RBAC) resources in one place and mirror your changes to follower clusters.

Scale your Sensu implementation

As the number of entities and checks in your Sensu implementation grows, so does the rate of events being written to the datastore. In clustered etcd deployments, each event must be replicated to each cluster member, which increases network and disk IO utilization.

Sensu's Enterprise datastore allows you to configure an external PostgreSQL instance for event storage so you can scale your monitoring and observability workflows beyond etcd's 8GB limit. Scale your Sensu implementation to many thousands of events per second, achieve much higher rates of event processing, and minimize the replication communication between etcd peers.

Read the datastore reference for the Enterprise datastore requirements and specifications.

For deployments at scale, configuration management tools can help ensure repeatable Sensu deployments and consistent configuration among Sensu backends. Ansible, Chef, and Puppet have well-defined Sensu modules to help you get started.

Hardware requirements

Sensu backend requirements

Backend minimum requirements

This configuration is the minimum required to run the Sensu backend (although it is insufficient for production use):

- 64-bit Intel or AMD CPU
- 4 GB RAM
- 4 GB free disk space
- 10 mbps network link

Review the backend recommended configuration for production recommendations.

Backend recommended configuration

This backend configuration is recommended as a baseline for production use to ensure a good user and operator experience:

- 64-bit four-core Intel or AMD CPU
- 8 GB RAM
- SSD [non-volatile memory express (NVMe) or serial advanced technology attachment 3 (SATA3)]
- Gigabit ethernet

Using additional resources (and even over-provisioning) further improves stability and scalability.

The Sensu backend is typically CPU- and storage-intensive. In general, the backend's use of these resources scales linearly with the total number of checks executed by all Sensu agents connecting to the backend.

The Sensu backend is a massively parallel application that can scale to any number of CPU cores. Provision approximately one CPU core for every 50 checks per second (including agent keepalives). For most installations, four CPU cores are sufficient. Larger installations may find that more CPU cores (8+) are necessary.

Every executed Sensu check results in storage writes. When provisioning storage, a good guideline is to have twice as many **sustained disk input/output operations per second (IOPS)** as you expect to have events per second.

Don't forget to include agent keepalives in your calculation. Each agent publishes a keepalive every 20 seconds. For example, in a cluster of 100 agents, you can expect the agents to consume 10 write IOPS for keepalives.

The Sensu backend uses a relatively modest amount of RAM in most circumstances. Larger production deployments use more RAM (8+ GB).

Sensu agent requirements

Agent minimum requirements

This configuration is the minimum required to run the Sensu agent (although it is insufficient for production use):

- 386, amd64, ARMv5, or MIPS CPU
- 128 MB RAM
- 10 mbps network link

Review the agent recommended configuration for production recommendations.

Agent recommended configuration

This agent configuration is recommended as a baseline for production use to ensure a good user and operator experience:

- 64-bit four-core Intel or AMD CPU
- 512 MB RAM

- Gigabit ethernet

The Sensu agent itself is lightweight and should be able to run on all but the most modest hardware. However, because the agent is responsible for executing checks, you should factor the agent's responsibilities into your hardware provisioning.

Networking recommendations

Sensu uses WebSockets for communication between the agent and backend. All communication occurs over a single TCP socket.

We recommend that you connect backends and agents via gigabit ethernet, but any reliable network link should work (for example, WiFi and 4G). If the backend logs include WebSocket timeouts, you may need to use a more reliable network link between the backend and agents.

Cloud recommendations

For all cloud providers, we recommend using local NVMe SSDs for storage and deploying all Sensu backends and etcd instances in the same region.

Sensu is compatible with all cloud provider database instances. We recommend using PostgreSQL with high availability for the event store.

> ***NOTE**: Sensu does not require a particular CPU manufacturer for cloud storage.*

Amazon EC2

For Sensu backends or etcd nodes, the recommended Amazon EC2 instance type and size is **M5d.xlarge**. The M5d.xlarge instance provides four vCPU, 16 GB of RAM, up to 10 gbps network connectivity, and a 150-GB NVMe SSD directly attached to the instance host, which is optimal for sustained disk IOPS.

Microsoft Azure

Use the **D4ds v4** Microsoft Azure virtual machine for Sensu backends or etcd nodes. The D4ds v4 virtual machine provides four vCPU, 16 GB of RAM, and a 150-GB SSD directly attached to the

instance host (optimal for sustained disk IOPS).

Digital Ocean

Use Digital Ocean Storage-Optimized Droplets for Sensu backends or etcd. The minimum Storage-Optimized Droplet plan provides two vCPU, 16 GB of RAM, and a 300-GB NVMe SSD. Storage is directly attached to the hypervisor rather than connected via network.

Google Cloud

For Sensu backends or etcd nodes, the recommended Google Cloud Compute Engine type and size is **n2-standard-4**, with SSD provisioned space. The n2-standard-4 compute engine provides four vCPU, 16 GB of RAM, and up to 10 gbps network connectivity.

Google Cloud offers disk space separately, and we recommend at least 150 GB of SSD provisioned space for Sensu backends running embedded etcd.

You can use Google Cloud's regional managed instance groups (MIGs) to deploy Sensu backends and etcd instances.

Install Sensu

This installation guide describes how to install the Sensu backend, Sensu agent, and sensuctl command line tool.

These instructions explain how to install Sensu for proof-of-concept purposes or testing in a development environment. We recommend using a supported package to follow this guide.

To build Sensu Go from source (OSS), follow the Sensu Go installation instructions on GitHub.

> ***NOTE**: If you're trying Sensu for the first time, consider following the the Sensu Go workshop instead. The workshop includes a local sandbox environment and a collection of resources designed to help new users learn and test Sensu.*
>
> *If you will deploy Sensu to your infrastructure, we recommend securing your installation with transport layer security (TLS) in addition to using one of our supported packages, Docker images, or configuration management integrations. Read Generate certificates next to get the certificates you will need for TLS.*

Sensu downloads are provided under the Sensu commercial license.

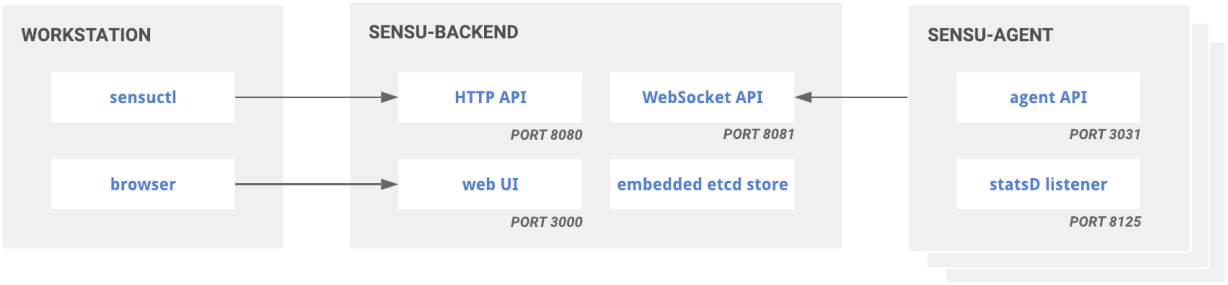
Sensu Go is packaged for Linux, Windows (agent and sensuctl only), macOS (sensuctl only), and Docker. Review supported platforms for more information.

Architecture overview

Sensu works differently from other monitoring and observability solutions. Instead of provisioning each device, server, container, or sidecar you want to monitor, you install the Sensu agent on each infrastructure component.

Sensu agents are lightweight clients that run on the infrastructure components you want to monitor. Agents are responsible for creating status and metric events to send to the Sensu backend event pipeline. Agents automatically register with Sensu as entities when you start them up and connect to the Sensu backend with no need for further provisioning. You only need to specify the IP address for the Sensu backend server — you do not need to list the components to monitor in the backend.

The **Sensu backend** is powered by an an embedded transport and etcd datastore. The backend sends specific checks for each agent to execute according to the subscriptions you define in the agent configuration. Sensu automatically downloads the files needed to run the checks from an asset repository like Bonsai or a local repo and schedules the checks on each agent. The agents execute the checks the backend sends to their subscriptions and send the resulting status and metric events to the backend event pipeline, which gives you flexible, automated workflows to route these events.



The Sensu backend keeps track of all self-registered agents. If the backend loses a keepalive signal from any of the agents, it flags the agent and generates an event. You can configure your instance so that when an agent (for example, a server) shuts down gracefully, the agent automatically de-registers from the backend and does not generate an alert.

Sensu backends require persistent storage for their embedded database, disk space for local asset caching, and several exposed ports. Agents that use Sensu dynamic runtime assets require some disk space for a local cache.

For more information, read Secure Sensu. Read deployment architecture and hardware requirements for deployment recommendations.

Ports

Sensu backends require the following ports:

Port	Protocol	Description
2379	gRPC	Sensu storage client: Required for Sensu backends using an external etcd instance
2380	gRPC	Sensu storage peer: Required for etcd cluster members to communicate directly with their peers
3000	HTTP/HT	Sensu web UI: Required for all Sensu backends using a

	TPS	Sensu web UI
6060	HTTP/HTTPS	Required for all Sensu backends when performance profiling is enabled via debug setting
8080	HTTP/HTTPS	Sensu API: Required for all users accessing the Sensu API
8081	WS/WSS	Agent API (backend WebSocket): Required for all Sensu agents connecting to a Sensu backend

The Sensu agent uses the following ports:

Port	Protocol	Description
3030	TCP/UDP	Sensu agent socket: Required for Sensu agents using the agent socket
3031	HTTP	Sensu agent API: Required for all users accessing the agent API
8125	UDP	StatsD listener: Required for all Sensu agents using the StatsD listener

The agent TCP and UDP sockets are deprecated in favor of the agent API.

Install the Sensu backend

The Sensu backend is available for Debian- and RHEL-family distributions and Docker.Review supported platforms for more information.

1. Download

DOCKER

```
# All Sensu Docker images contain a Sensu backend and a Sensu agent

# Pull the Alpine-based image
docker pull sensu/sensu
```

```
# Pull the image based on Red Hat Enterprise Linux
docker pull sensu/sensu-rhel
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh |
sudo bash

# Install the sensu-go-backend package
sudo apt-get install sensu-go-backend
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh |
sudo bash

# Install the sensu-go-backend package
sudo yum install sensu-go-backend
```

2. Configure and start

You can configure the Sensu backend with `sensu-backend start` flags (recommended) or an `/etc/sensu/backend.yml` file. The Sensu backend requires the `state-dir` flag at minimum, but other useful configurations and templates are available.

> ***NOTE**: If you are using Docker, intitialization is included in this step when you start the backend rather than in 3. Initialize. For details about intialization in Docker, read the backend reference.*

DOCKER

```
# Replace `<username>` and `<password>` with the username and password
# you want to use for your admin user credentials.
docker run -v /var/lib/sensu:/var/lib/sensu \
-d --name sensu-backend \
-p 3000:3000 -p 8080:8080 -p 8081:8081 \
-e SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username> \
-e SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password> \
```

```
sensu/sensu:latest \
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

DOCKER

```
# Replace `<username>` and `<password>` with the username and password
# you want to use for your admin user credentials.
---
version: "3"
services:
  sensu-backend:
    ports:
    - 3000:3000
    - 8080:8080
    - 8081:8081
    volumes:
    - "sensu-backend-data:/var/lib/sensu/sensu-backend/etcd"
    command: "sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-
level debug"
    environment:
    - SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
    - SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
    image: sensu/sensu:latest

volumes:
  sensu-backend-data:
    driver: local
```

SHELL

```
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/backend.yml -o
/etc/sensu/backend.yml

# Start sensu-backend using a service manager
sudo systemctl start sensu-backend

# Verify that the backend is running
sudo systemctl status sensu-backend
```

SHELL

```
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/backend.yml -o
/etc/sensu/backend.yml

# Start sensu-backend using a service manager
sudo systemctl start sensu-backend

# Verify that the backend is running
sudo systemctl status sensu-backend
```

The backend reference includes a complete list of configuration options and backend initialization details.

> ***WARNING**: If you plan to run a Sensu cluster, make sure that each of your backend nodes is configured, running, and a member of the cluster before you continue the installation process.*

3. Initialize

> ***NOTE**: If you are using Docker, you already completed intitialization in 2. Configure and start. Skip ahead to 4. Open the web UI to continue the backend installation process. If you did not use environment variables to override the default admin credentials in step 2, skip ahead to Install sensuctl so you can change your default admin password immediately.*

With the backend running, run `sensu-backend init` to set up your Sensu administrator username and password. In this initialization step, you only need to set environment variables with a username and password string — no need for role-based access control (RBAC).

Replace `<username>` and `<password>` with the username and password you want to use:

SHELL

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
sensu-backend init
```

SHELL

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
sensu-backend init
```

For details about initializing the Sensu backend, read the backend reference.

***NOTE**: You may need to allow access to the ports Sensu requires in your local server firewall. Refer to the documentation for your operating system to configure port access as needed.*

4. Open the web UI

***COMMERCIAL FEATURE**: Access the Sensu web UI in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The web UI provides a unified view of your observability events and user-friendly tools to reduce alert fatigue. After starting the Sensu backend, open the web UI by visiting http://localhost:3000. You may need to replace `localhost` with the hostname or IP address where the Sensu backend is running.

To log in to the web UI, enter your Sensu user credentials. If you are using Docker and you did not specify environment variables to override the default admin credentials, your user credentials are username `admin` and password `P@ssw0rd!`. Otherwise, your user credentials are the username and password you provided with the `SENSU_BACKEND_CLUSTER_ADMIN_USERNAME` and `SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD` environment variables.

Select the ☰ icon to explore the web UI.

5. Make a request to the /health API

To make sure the backend is up and running, use the Sensu /health API to check the backend's health. You should receive a response that includes `"Healthy": true`.

```
curl http://127.0.0.1:8080/health
```

Now that you've installed the Sensu backend, install and configure sensuctl to connect to your backend URL. Then you can install a Sensu agent and start monitoring your infrastructure.

Install sensuctl

Sensuctl is a command line tool for managing resources within Sensu. It works by calling Sensu's HTTP API to create, read, update, and delete resources, events, and entities. Sensuctl is available for Linux, Windows, and macOS.

To install sensuctl:

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh |
sudo bash

# Install the sensu-go-cli package
sudo apt-get install sensu-go-cli
```

SHELL

```
# Add the Sensu repository
curl https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh | sudo
bash

# Install the sensu-go-cli package
sudo yum install sensu-go-cli
```

POWERSHELL

```
# Download sensuctl for Windows amd64
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-
go_6.8.2_windows_amd64.zip  -OutFile C:\Users\Administrator\sensu-
go_6.8.2_windows_amd64.zip

# Or for Windows 386
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-
go_6.8.2_windows_386.zip  -OutFile C:\Users\Administrator\sensu-
go_6.8.2_windows_386.zip
```

```
# Unzip the file with PowerShell for Windows amd64
Expand-Archive -LiteralPath 'C:\Users\Administrator\sensu-
go_6.8.2_windows_amd64.zip' -DestinationPath 'C:\\Program Files\sensu\sensuctl\bin'

# or for Windows 386
Expand-Archive -LiteralPath 'C:\Users\Administrator\sensu-go_6.8.2_windows_386.zip'
-DestinationPath 'C:\\Program Files\sensu\sensuctl\bin'
```

SHELL

```
# Download the latest release
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-
go_6.8.2_darwin_amd64.tar.gz

# Extract the archive
tar -xvf sensu-go_6.8.2_darwin_amd64.tar.gz

# Copy the executable into your PATH
sudo cp sensuctl /usr/local/bin/
```

To start using sensuctl, run `sensuctl configure` and log in with your user credentials, namespace, and Sensu backend URL. To configure sensuctl using default values:

```
sensuctl configure -n \
--username 'YOUR_USERNAME' \
--password 'YOUR_PASSWORD' \
--namespace default \
--url 'http://127.0.0.1:8080'
```

Here, the `-n` flag triggers non-interactive mode. Run `sensuctl config view` to view your user profile

For more information about sensuctl, read the sensuctl documentation.

Change default admin password

If you are using Docker and you did not use environment variables to override the default admin

credentials in step 2 of the backend installation process, we recommend that you change the default admin password as soon as you have installed sensuctl.Run:

```
sensuctl user change-password --interactive
```

Install Sensu agents

The Sensu agent is available for Debian- and RHEL-family distributions, Windows, and Docker.Review supported platforms for more information.

1. Download

DOCKER

```
# All Sensu images contain a Sensu backend and a Sensu agent

# Pull the Alpine-based image
docker pull sensu/sensu

# Pull the RHEL-based image
docker pull sensu/sensu-rhel
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh |
sudo bash

# Install the sensu-go-agent package
sudo apt-get install sensu-go-agent
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh |
sudo bash

# Install the sensu-go-agent package
```

```
sudo yum install sensu-go-agent
```

POWERSHELL

```
# Download the Sensu agent for Windows amd64
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go-agent_6.8.2.6788_en-US.x64.msi  -OutFile "$env:userprofile\sensu-go-agent_6.8.2.6788_en-US.x64.msi"

# Or for Windows 386
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.8.2/sensu-go-agent_6.8.2.6788_en-US.x86.msi  -OutFile "$env:userprofile\sensu-go-agent_6.8.2.6788_en-US.x86.msi"

# Install the Sensu agent for Windows amd64
msiexec.exe /i $env:userprofile\sensu-go-agent_6.8.2.6788_en-US.x64.msi /qn

# Or for Windows 386
msiexec.exe /i $env:userprofile\sensu-go-agent_6.8.2.6788_en-US.x86.msi /qn

# Or via Chocolatey
choco install sensu-agent
```

2. Configure and start

You can configure the Sensu agent with `sensu-agent start` flags (recommended) or an `/etc/sensu/agent.yml` file. The Sensu agent requires the `--backend-url` flag at minimum, but other useful configurations and templates are available.

DOCKER

```
# If you are running the agent locally on the same system as the Sensu backend,
# add `--link sensu-backend` to your Docker arguments and change the backend
# URL to `--backend-url ws://sensu-backend:8081`.

# Start an agent with the system subscription
docker run -v /var/lib/sensu:/var/lib/sensu -d \
--name sensu-agent sensu/sensu:latest \
sensu-agent start --backend-url ws://sensu.yourdomain.com:8081 --log-level debug --subscriptions system --api-host 0.0.0.0 --cache-dir /var/lib/sensu
```

DOCKER

```
# Start an agent with the system subscription
---
version: "3"
services:
  sensu-agent:
    image: sensu/sensu:latest
    ports:
    - 3031:3031
    volumes:
    - "sensu-agent-data:/var/lib/sensu"
    command: "sensu-agent start --backend-url ws://sensu-backend:8081 --log-level
debug --subscriptions system --api-host 0.0.0.0 --cache-dir /var/lib/sensu"

volumes:
  sensu-agent-data:
    driver: local
```

SHELL

```
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/agent.yml -o
/etc/sensu/agent.yml

# Start sensu-agent using a service manager
sudo systemctl start sensu-agent
```

SHELL

```
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/agent.yml -o
/etc/sensu/agent.yml

# Start sensu-agent using a service manager
sudo systemctl start sensu-agent
```

POWERSHELL

```
# Copy the example agent config file from
%ALLUSERSPROFILE%\sensu\config\agent.yml.example
# (default: C:\ProgramData\sensu\config\agent.yml.example) to
C:\ProgramData\sensu\config\agent.yml
cp C:\ProgramData\sensu\config\agent.yml.example C:\ProgramData\sensu\config\agent.yml

# Change to the sensu\sensu-agent\bin directory where you installed Sensu
cd 'C:\Program Files\sensu\sensu-agent\bin'

# Run the sensu-agent executable
./sensu-agent.exe

# Install and start the agent
./sensu-agent service install
```

The agent reference includes a complete list of configuration options.

3. Verify keepalive events

Sensu keepalives are the heartbeat mechanism used to ensure that all registered agents are operating and can reach the Sensu backend. To confirm that the agent is registered with Sensu and is sending keepalive events, open the entity page in the Sensu web UI or run `sensuctl entity list`.

4. Verify an example event

With your backend and agent still running, send this request to the Sensu core/v2/events API:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "check-mysql-status"
    },
    "status": 1,
    "output": "could not connect to mysql"
  }
}' \
```

```
http://127.0.0.1:3031/events
```

This request creates a `warning` event that you can view in your web UI Events page.

To create an OK event, change the `status` to `0` and resend. You can change the `output` value to `connected to mysql` to use a different message for the OK event.

Next steps

Now that you have installed Sensu, you're ready to build your observability pipelines! Here are some ideas for next steps.

Get started with Sensu

If you're ready to try Sensu, one of these pathways can get you started:

- Manually trigger an event that sends alerts to your email inbox.
- Create a check to monitor CPU usage and send Slack alerts based on your check.
- Collect metrics with a Sensu check and use a handler to populate metrics in InfluxDB.
- Use the sensuctl dump command to export all of your events and resources as a backup — then use sensuctl create to restore if needed.

Deploy Sensu outside your local development environment

To deploy Sensu for use outside of a local development environment, first decide whether you want to run a Sensu cluster. A Sensu cluster is a group of three or more sensu-backend nodes, each connected to a shared database provided either by Sensu's embedded etcd or an external etcd cluster.

Clustering allows you to absorb the loss of a backend node, prevent data loss, and distribute the network load of agents. However, scaling a single backend to a cluster or migrating a cluster from cleartext HTTP to encrypted HTTPS without downtime can require a number of tedious steps. For this reason, we recommend that you decide whether your deployment will require clustering as part of your initial planning effort.

No matter whether you deploy a single backend or a clustered configuration, begin by securing Sensu with transport layer security (TLS). The first step in setting up TLS is to generate the certificates you

need. Then, follow our Secure Sensu guide to make Sensu production-ready.

After you've secured Sensu, read Run a Sensu cluster if you are setting up a clustered configuration.

Commercial features

Sensu Inc. offers support packages for Sensu Go and commercial features designed for monitoring at scale.

All commercial features are free for your first 100 entities. To learn more about Sensu Go commercial licenses for more than 100 entities, contact the Sensu sales team.

If you already have a Sensu commercial license, log in to your Sensu account and download your license file. Save your license to a file such as `sensu_license.yml` or `sensu_license.json`.

Use sensuctl to activate your license:

SHELL

```
sensuctl create --file sensu_license.yml
```

SHELL

```
sensuctl create --file sensu_license.json
```

You can use sensuctl to view your license details at any time.

```
sensuctl license info
```

Deployment architecture for Sensu

This guide describes various planning considerations and recommendations for a production-ready Sensu deployment, including details related to communication security and common deployment architectures.

etcd is a key-value store that is used by applications of varying complexity, from simple web apps to Kubernetes. The Sensu backend uses an embedded etcd instance for storing both configuration and observability event data, so you can get Sensu up and running without external dependencies.

By building atop etcd, Sensu's backend inherits a number of characteristics to consider when you're planning for a Sensu deployment.

Create and maintain clusters

Sensu's embedded etcd supports initial cluster creation via a static list of peer URLs. After you create a cluster, you can add and remove cluster members with etcdctl tooling.

If you have a healthy clustered backend, you only need to make Sensu API calls to any one of the cluster members. The cluster protocol will replicate your changes to all cluster members.

Read Run a Sensu cluster and the etcd documentation for more information.

Hardware sizing

Because etcd's design prioritizes consistency across a cluster, the speed with which write operations can be completed is very important to the Sensu cluster's performance and health. This means that you should provision Sensu backend infrastructure to provide sustained input/output operations per second (IOPS) appropriate for the rate of observability events the system will be required to process.

To maximize Sensu Go performance, we recommend that you:

- Follow our recommended backend hardware configuration.
- Implement documented etcd system tuning practices.
-

- Benchmark your etcd storage volume to establish baseline IOPS for your system.
- Scale event storage using PostgreSQL to reduce the overall volume of etcd transactions.

Communications security

Whether you're using using a single Sensu backend or multiple Sensu backends in a cluster, communication with the backend's various network ports (web UI, HTTP API, WebSocket API, etcd client and peer) occurs in cleartext by default. We recommend that you encrypt network communications via TLS, which requires planning and explicit configuration.

Plan TLS for etcd

The URLs for each member of an etcd cluster are persisted to the database after initialization. As a result, moving a cluster from cleartext to encrypted communications requires resetting the cluster, which destroys all configuration and event data in the database. Therefore, we recommend planning for encryption before initiating a clustered Sensu backend deployment.

***WARNING**: Reconfiguring a Sensu cluster for TLS post-deployment will require resetting all etcd cluster members, resulting in the loss of all data.*

As described in Secure Sensu, the backend uses a shared certificate and key for web UI and agent communications. You can secure communications with etcd using the same certificate and key. The certificate's Common Name (CN) or Subject Alternative Name (SAN) must include the network interfaces and DNS names that will point to those systems.

***NOTE**: As of Go 1.15, certificates must include their CN as an SAN field. To prevent connection errors, follow Generate certificates to make sure your certificates' SAN fields include their CNs.*

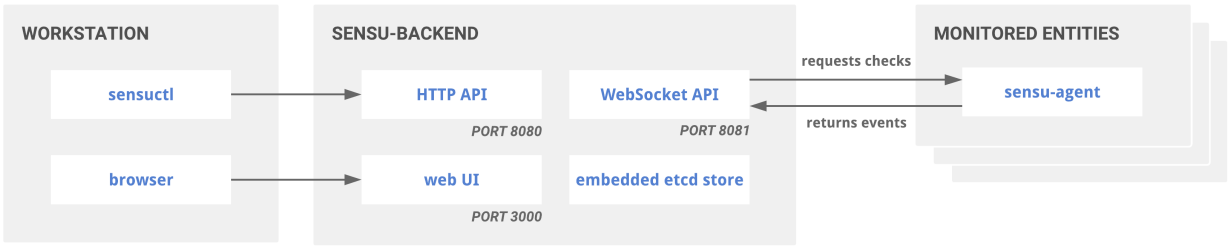
Read Run a Sensu cluster and the etcd documentation for more information about TLS setup and configuration, including a walkthrough for generating TLS certificates for your cluster.

Common Sensu architectures

Depending on your infrastructure and the type of environments you'll be monitoring, you may use one or a combination of these architectures to best fit your needs.

Single backend (standalone)

The single backend (standalone) with embedded etcd architecture requires minimal resources but provides no redundancy in the event of failure.



Single Sensu Go backend or standalone architecture

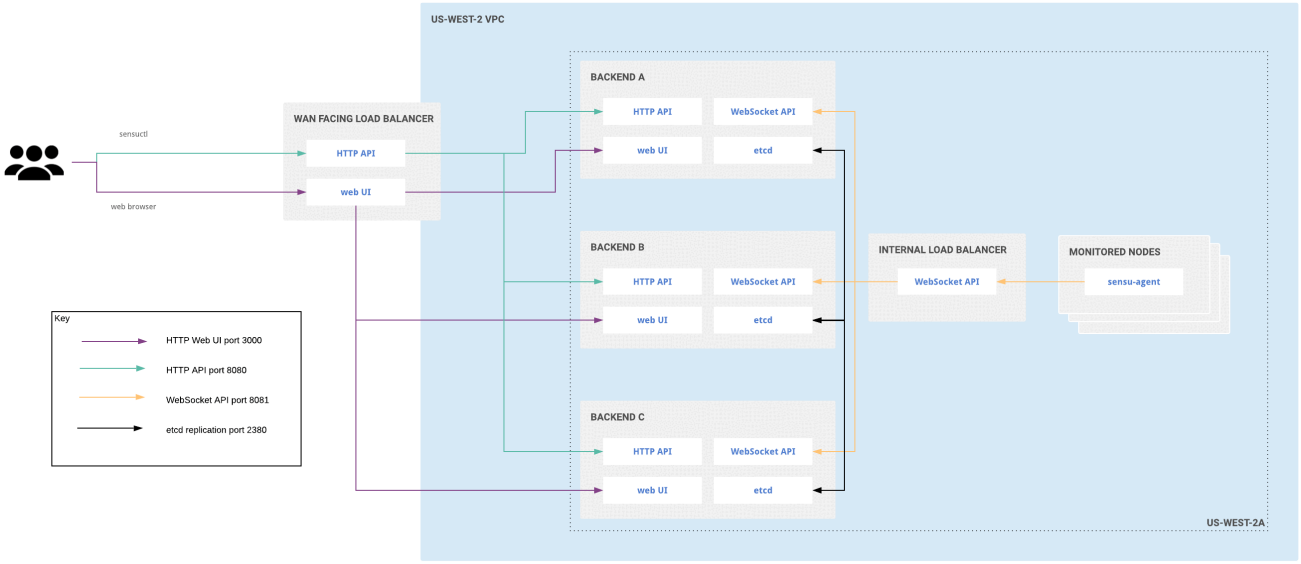
You can reconfigure a single backend as a member of a cluster, but this operation requires destroying the existing database.

The single backend (standalone) architecture may be a good fit for small- to medium-sized deployments (such as monitoring a remote office or datacenter), deploying alongside individual auto-scaling groups, or deploying in various segments of a logical environment spanning multiple cloud providers.

For example, in environments with unreliable WAN connectivity, having agents connect to a local backend may be more reliable than having agents connect over WAN or VPN tunnel to a backend running in a central location.

Clustered deployment for single availability zone

To increase availability and replicate both configuration and data, join the embedded etcd databases of multiple Sensu backend instances together in a cluster.Read Run a Sensu cluster for more information.

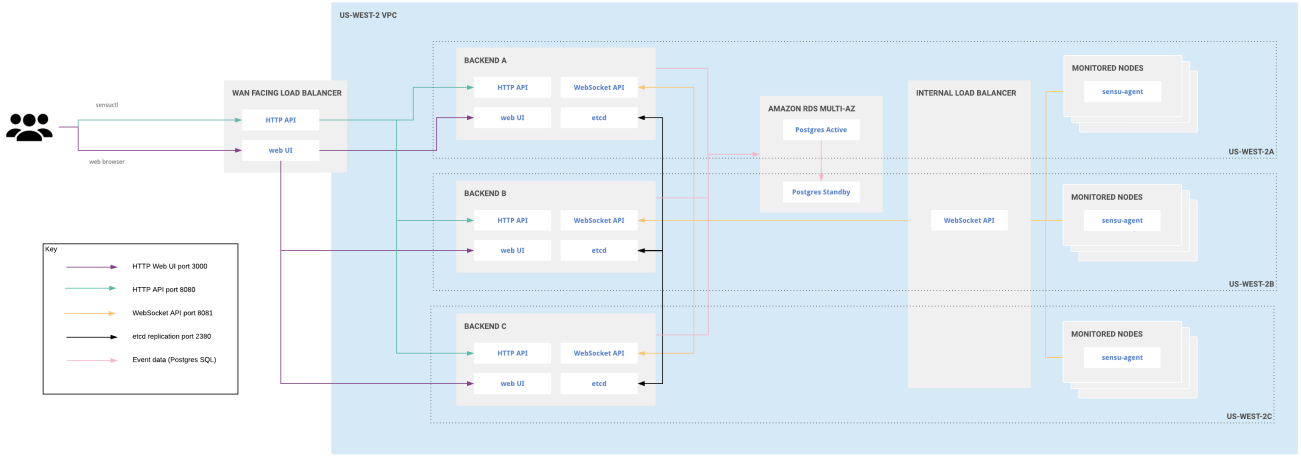


Clustered Sensu Go architecture for a single availability zone

Clustering requires an odd number of backend instances. Although larger clusters provide better fault tolerance, write performance suffers because data must be replicated across more machines. The etcd maintainers recommend clusters of 3, 5, or 7 backends. Read the etcd documentation for more information.

Clustered deployment for multiple availability zones

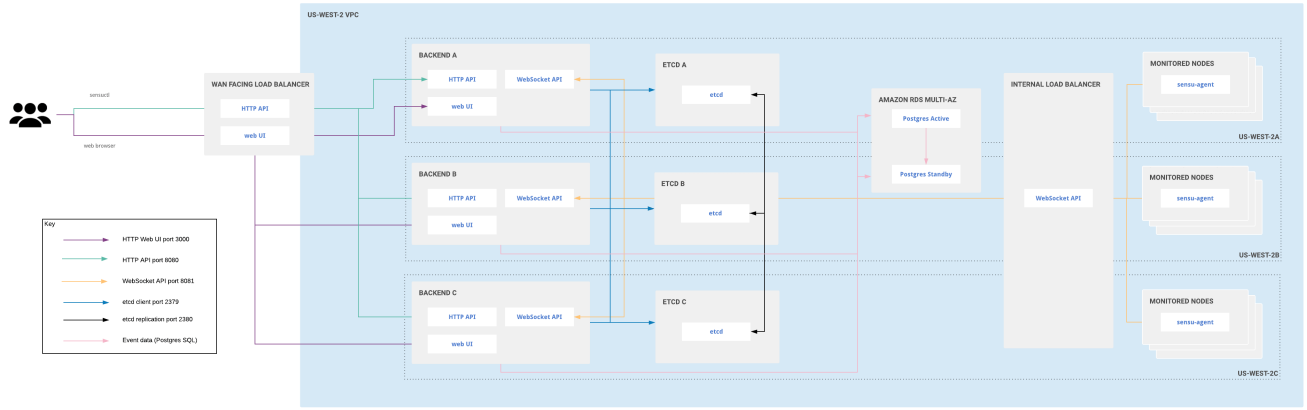
Distributing infrastructure across multiple availability zones in a given region helps ensure continuous availability of customer infrastructure in the region if any one availability zone becomes unavailable. With this in mind, you can deploy a Sensu cluster across multiple availability zones in a given region, configured to tolerate reasonable latency between those availability zones.



Clustered Sensu Go architecture for multiple availability zones

Large-scale clustered deployment for multiple availability zones

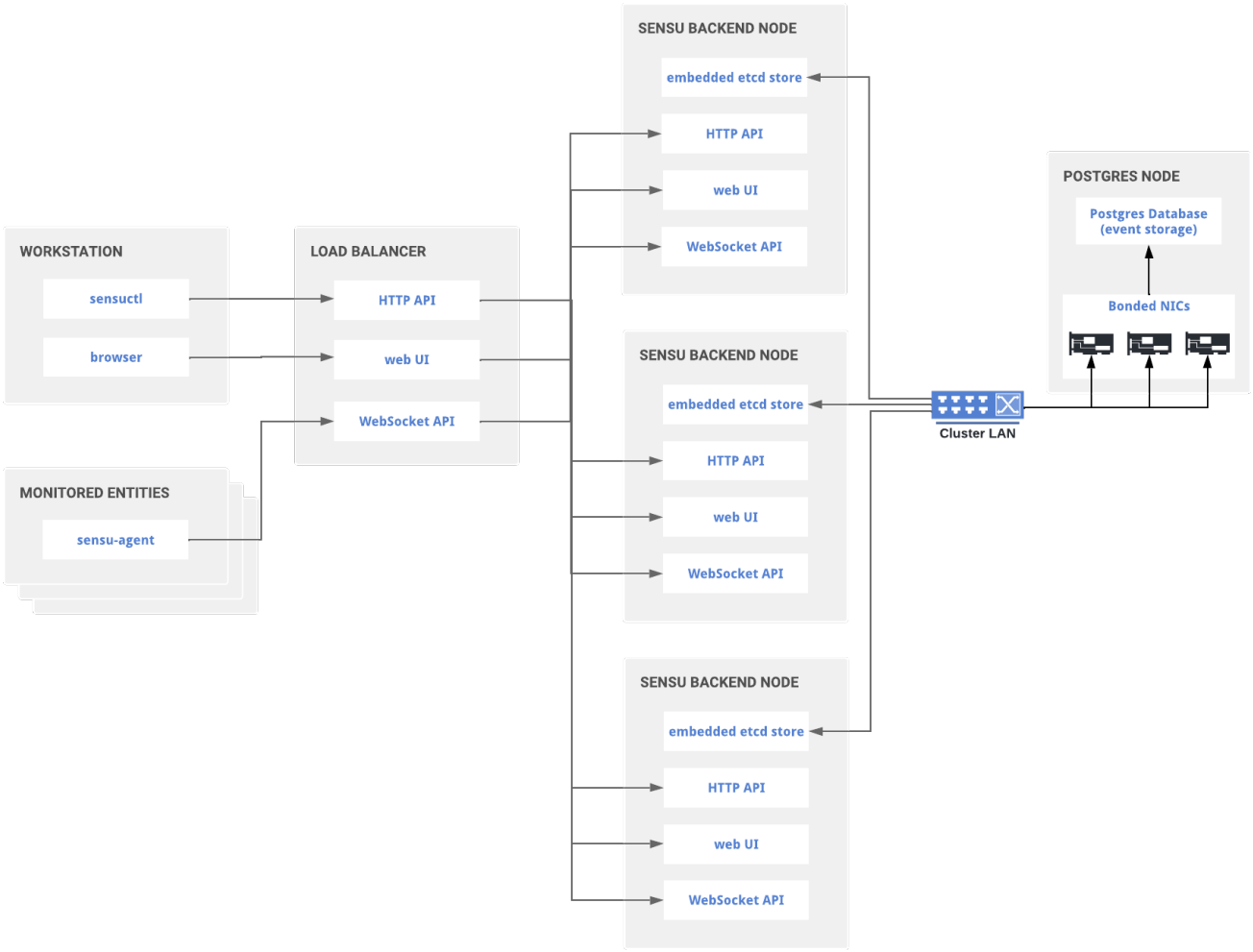
In a large-scale clustered Sensu Go deployment, you can use as many backends as you wish. Use one etcd node per availiability zone, with a minimum of three etcd nodes and a maximum of five. Three etcd nodes allow you to tolerate the loss of a single node with minimal effect on performance. Five etcd nodes allow you to tolerate the loss of two nodes, but with a greater effect on performance.



Large-scale clustered Sensu Go architecture for multiple availability zones

Scaled cluster performance with PostgreSQL

To achieve the high rate of event processing that many enterprises require, Sensu supports PostgreSQL event storage as a commercial feature. Read the datastore reference to configure the Sensu backend to use PostgreSQL for event storage.



Clustered Sensu Go architecture with PostgreSQL event storage

In load testing, Sensu Go has proven capable of processing 36,000 events per second when using PostgreSQL as the event store. Review the sensu-perf project repository for a detailed explanation of our testing methodology and results.

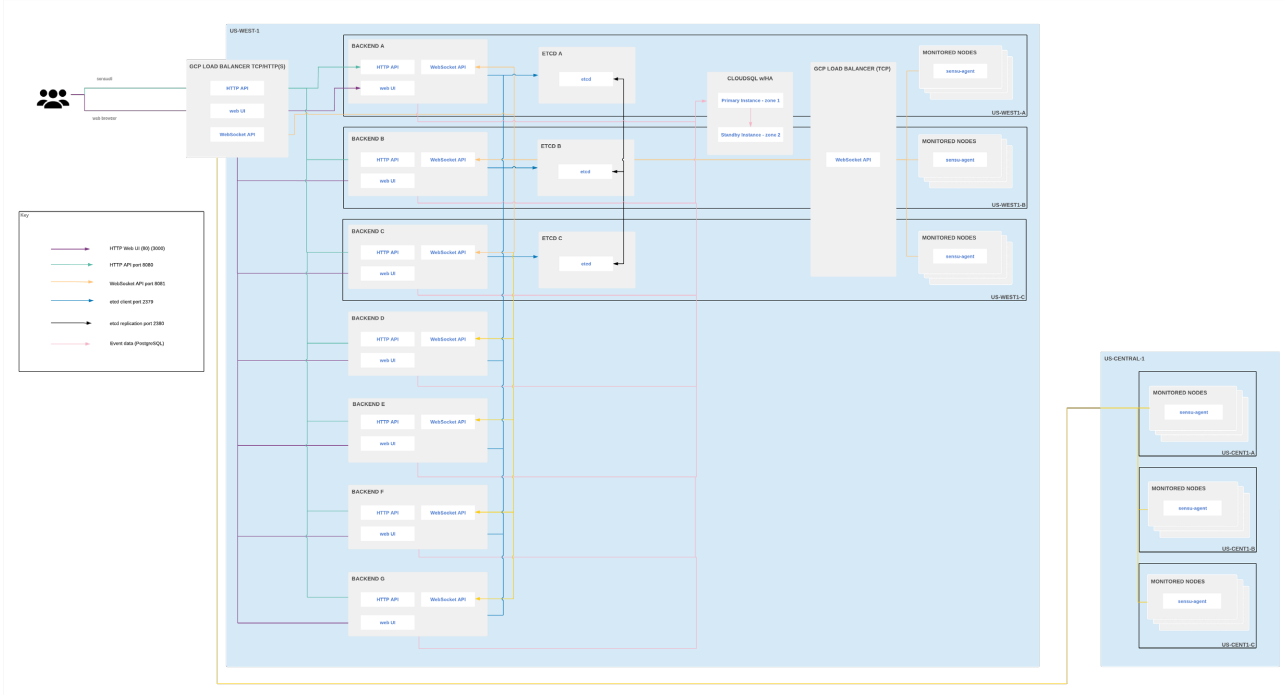
Large-scale cloud deployment for multiple regions

In a large-scale cloud deployment for multiple regions, place all backends, etcd nodes, and event datastores in a single region. The backends communicate with agents in other regions via WebSocket transport. This configuration allows you to load-balance traffic between the backends in the main regions and the agents in other regions.

> ***NOTE**: Do not use read replicas in a cloud deployment. Sensu is write-heavy, and the brief, unavoidable replication delays will cause inconsistency between etcd data and PostgreSQL data.*

This diagram depicts an example architecture for a Google Cloud Platform (GCP) deployment, but you

can reproduce this architecture with your preferred cloud provider:



Example Sensu Go architecture for multi-region cloud deployments

In this example, the load balancer translates traffic on port 80 to port 3000 so that users do not need to include `:3000` in the web UI URL. API traffic is load-balanced to the backends as well.

Architecture considerations

Networking

Clustered deployments benefit from a fast and reliable network. Ideally, they should be co-located in the same network segment with as little latency as possible between all the nodes. We do not recommend clustering backends across disparate subnets or WAN connections.

Although 1GbE is sufficient for common deployments, larger deployments will benefit from 10GbE, which allows a shorter mean time to recovery.

As the number of agents connected to a backend cluster grows, so will the amount of communication between members of the cluster required for data replication. With this in mind, clusters with a thousand or more agents should use a discrete network interface for peer communication.

Load balancing

Although you can configure each Sensu agent with the URLs for multiple backend instances, we recommend that you configure agents to connect to a load balancer. This approach gives operators more control over agent connection distribution and makes it possible to replace members of the backend cluster without updates to agent configuration.

Conversely, you cannot configure the sensuctl command line tool with multiple backend URLs. Under normal conditions, sensuctl communications and browser access to the web UI should be routed via a load balancer.

Load balancing algorithms

If the load balancer uses round robin mode, when an agent comes online, the load balancer sends the agent's traffic to whichever backend is next in the pattern, regardless of load. This can result in slow load balancing among backends, especially after restarting a backend.

In least connection mode, the load balancer sends new agent traffic to the backend with the least traffic. This helps to evenly distribute the load among backends more quickly.

Configuration management

We recommend using configuration management tools to deploy Sensu in production and at scale.

- Pin versions of Sensu-related software to ensure repeatable Sensu deployments.
- Ensure consistent configuration between Sensu backends.

The configuration management tools listed here have well-defined Sensu modules to help you get started.

Ansible

The Ansible role to deploy and manage Sensu Go is available in the Sensu Go Ansible Collection.

The Sensu Go Ansible Collection documentation site includes installation instructions, example playbooks, and module references.

Chef

The Chef cookbook for installing and configuring Sensu is available in the Sensu Go Chef Cookbook.

Contact us for more information about Sensu + Chef.

Puppet

The Puppet module to install Sensu is available in the Sensu Puppet Module.

Sensu partnered with Tailored Automation to enhance the Puppet module with new features and bug fixes.

Generate certificates for your Sensu installation

This guide explains how to generate the certificates you need to secure a Sensu cluster and its agents.

When deploying Sensu for use outside of a local development environment, you should secure it using transport layer security (TLS). TLS uses encryption to provide security for communication between Sensu backends and agents as well as communication between human operators and the Sensu backend, such as web UI or sensuctl access.

Because reconfiguring an existing Sensu deployment from cleartext to TLS can be time-consuming, we recommend that you configure TLS for your backend from the very beginning.

TLS is also required to use some of Sensu's commercial features, like secrets management and mutual TLS authentication (mTLS).

Prerequisites

To use this guide, you must have already installed Sensu on:

- One backend system or three backend systems that you plan to cluster together.
- One or more agents.

Public key infrastructure (PKI)

To use TLS, you must either possess existing public key infrastructure (PKI) or generate your own Certificate Authority (CA) for issuing certificates.

This guide describes how to set up a minimal CA and generate the certificates you need to secure Sensu communications for a clustered backend and agents.

If your organization has existing PKI for certificate issuance, you can adapt the suggestions in this guide to your organization's PKI. Recommended practices for deploying and maintaining production PKI can be complex and case-specific, so they are not included in the scope of this guide.

Issue certificates

Use a CA certificate and key to generate certificates and keys to use with Sensu backends and agents.

This guide uses the CloudFlare cfssl toolkit to generate a CA and self-signed certificates from that CA. The examples assume that you'll install the certificates and keys in the `/etc/sensu/tls` directory.

Install TLS

The CloudFlare cfssl toolkit is released as a collection of command-line tools.

These tools only need to be installed on one system to generate your CA and issue certificates.

You may install the toolkit on your laptop or workstation and store the files there for safekeeping or install the toolkit on one of the systems where you'll run the Sensu backend. The example in this guide installs cfssl on a Linux system.

1. Download the cfssl executable:

```
sudo curl -L
https://github.com/cloudflare/cfssl/releases/download/v1.4.1/cfssl_1.4.1_linux_
amd64 -o /usr/local/bin/cfssl
```

2. Download the cfssljson executable:

```
sudo curl -L
https://github.com/cloudflare/cfssl/releases/download/v1.4.1/cfssljson_1.4.1_li
nux_amd64 -o /usr/local/bin/cfssljson
```

3. Install the cfssl and cfssljson executables in /usr/local/bin:

```
sudo chmod +x /usr/local/bin/cfssl*
```

4. Verify the cfssl executable is version 1.4.1 and runtime go1.12.12:

```
cfssl version
```

5. Verify the cfssljson executable is version 1.4.1 and runtime go1.12.12:

```
cfssljson -version
```

Create a Certificate Authority (CA)

Follow these steps to create a CA with cfssl and cfssljson:

1. Create `/etc/sensu/tls` (which does not exist by default):

```
mkdir -p /etc/sensu/tls
```

2. Navigate to the new `/etc/sensu/tls` directory:

```
cd /etc/sensu/tls
```

3. Create the CA:

```
echo '{"CN":"Sensu Test CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert
-initca - | cfssljson -bare ca -
```

4. Define signing parameters and profiles (the agent profile provides the “client auth” usage required for mTLS):

```
echo '{"signing":{"default":{"expiry":"17520h","usages":["signing","key
encipherment","client auth"]},"profiles":{"backend":{"usages":["signing","key
```

```
encipherment","server auth","client auth"],"expiry":"4320h"},"agent":
{"usages":["signing","key encipherment","client auth"],"expiry":"4320h"}}}}' >
ca-config.json
```

***NOTE**: We suggest a 6-month expiry duration for security, but you can use any duration you prefer when you define the `expiry` attribute value in the signing parameters.*

You should now have a directory at `/etc/sensu/tls` that contains the following files:

filename	description
`ca.pem`	CA root certificate. Required for all systems running the Sensu backend or agent. The agent and backend use `ca.pem` to validate server certificates at connection time.
`ca-key.pem`	CA root certificate private key.
`ca-config.json`	CA signing parameters and profiles. Not used by Sensu.
`ca.csr`	Certificate signing request for the CA root certificate. Not used by Sensu.

Generate backend cluster certificates

Now that you've generated a CA, you will use it to generate certificates and keys for each backend server (etcd peer).

For each backend server, document the IP addresses and hostnames to use in backend and agent communications.During initial configuration of a cluster of Sensu backends, you must describe every member of the cluster with a URL passed as the value of the `etcd-initial-cluster` parameter.

In issuing certificates for cluster members, the IP address or hostname used in these URLs must be represented in either the Common Name (CN) or Subject Alternative Name (SAN) records in the certificate.

***NOTE**: As of Go 1.15, certificates must include their CN as an SAN field.Follow the instructions in this guide to make sure your certificates' SAN fields include their CNs.*

This guide assumes a scenario with three backend members that are reachable via a `10.0.0.x` IP address, a fully qualified name (for example, `backend-1.example.com`), and an unqualified name (for example, `backend-1`):

Unqualified name	IP address	Fully qualified domain name (FQDN)	Additional names
backend-1	10.0.0.1	backend-1.example.com	localhost, 127.0.0.1
backend-2	10.0.0.2	backend-2.example.com	localhost, 127.0.0.1
backend-3	10.0.0.3	backend-3.example.com	localhost, 127.0.0.1

The additional names for localhost and 127.0.0.1 are added here for convenience and are not strictly required.

Use these name and address details to create two `*.pem` files and one `*.csr` file for each backend.

- The values provided for the ADDRESS variable will be used to populate the certificate's SAN records. For systems with multiple hostnames and IP addresses, add each to the comma-delimited value of the ADDRESS variable.
- The value provided for the NAME variable will be used to populate the certificate's CN record. It will also be used in the names for the `*.pem` and `*.csr` files.

For example, to create certificate and key files for the three backends:

backend-1

```
export ADDRESS=localhost,127.0.0.1,10.0.0.1,backend-1
export NAME=backend-1.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="backend" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

backend-2

```
export ADDRESS=localhost,127.0.0.1,10.0.0.2,backend-2
```

```
export NAME=backend-2.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="backend" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

backend-3

```
export ADDRESS=localhost,127.0.0.1,10.0.0.3,backend-3
export NAME=backend-3.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="backend" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

The `/etc/sensu/tls` directory should now include three files for each backend, in addition to the four original CA files:

filename	description	required on backend?
`backend-*.pem`	Backend server certificate	✅
`backend-*-key.pem`	Backend server private key	✅
`backend-*.csr`	Certificate signing request	

In our example with three backends, the directory listing for `/etc/sensu/tls` would include 13 files:

```
/etc/sensu/tls/
├── backend-1.example.com-key.pem
├── backend-1.example.com.pem
├── backend-1.example.com.csr
├── backend-2-key.example.com.pem
├── backend-2.example.com.pem
├── backend-2.example.com.csr
├── backend-3-key.example.com.pem
├── backend-3.example.com.pem
├── backend-3.example.com.csr
├── ca.pem
```

```
├── ca-key.pem
├── ca-config.json
└── ca.csr
```

***WARNING**: If you are **not** setting up agent mTLS authentication, delete the `ca-key.pem` file from the `/etc/sensu/tls` directory. The `ca-key.pem` file contains sensitive information and is no longer needed unless you are setting up agent mTLS authentication.*

To make sure the backend files in `/etc/sensu/tls` are accessible only by the `sensu` user, run:

```
chown sensu /etc/sensu/tls/*.pem
```

And:

```
chmod 400 /etc/sensu/tls/*.pem
```

Generate agent certificate

***NOTE**: Agent certificates are only required for agent mTLS authentication. If you are not configuring mTLS for Sensu agents, you do not need to generate agent certificates.*

Now you will generate a certificate that agents can use to connect to the Sensu backend. Sensu's commercial distribution offers support for authenticating agents via TLS certificates instead of a username and password.

For this certificate, you only need to specify a CN (here, `agent`) — you don't need to specify an address. You will create the files `agent.pem`, `agent-key.pem`, and `agent.csr`:

```
export NAME=agent
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="" -
profile=agent - | cfssljson -bare $NAME
```

The `/etc/sensu/tls` directory should now include a set of files for use by Sensu agents:

filename	description	required on agent?
`agent.pem`	Agent certificate	✅
`agent-key.pem`	Agent private key	✅
`agent.csr`	Certificate signing request	

***WARNING**: Before you continue, delete the `ca-key.pem` file from the `/etc/sensu/tls` directory. This file contains sensitive information and is no longer needed.*

To continue the example with three backends, the directory listing for `/etc/sensu/tls` will include 15 files after deleting the `ca-key.pem` file:

```
/etc/sensu/tls/
├── agent-key.pem
├── agent.pem
├── agent.csr
├── backend-1.example.com-key.pem
├── backend-1.example.com.pem
├── backend-1.example.com.csr
├── backend-2-key.example.com.pem
├── backend-2.example.com.pem
├── backend-2.example.com.csr
├── backend-3-key.example.com.pem
├── backend-3.example.com.pem
├── backend-3.example.com.csr
├── ca.pem
├── ca-config.json
└── ca.csr
```

To make sure the agent `/etc/sensu/tls` files are accessible only by the `sensu` user, run:

```
chown sensu /etc/sensu/tls/*.pem
```

And:

```
chmod 400 /etc/sensu/tls/*.pem
```

Install CA certificates

Before you install the CA certificates, **make sure that the `/etc/sensu/tls` directory does not contain the `ca-key.pem` file.** The `ca-key.pem` file contains sensitive information that is no longer needed, so you should delete it.

Also, make sure that `/etc/sensu/tls` includes the CA root certificate and key, as well as a certificate and key for each backend and agent you are securing.

We recommend installing the CA root certificate in the trust store of both your Sensu systems and those systems used by operators to manage Sensu. Installing the CA certificate in the trust store for these systems makes it easier to connect via web UI or sensuctl without being prompted to accept certificates signed by your self-generated CA.

SHELL

```
chmod 644 /etc/sensu/tls/ca.pem
chown root /etc/sensu/tls/ca.pem
sudo apt-get install ca-certificates -y
sudo ln -sfv /etc/sensu/tls/ca.pem /usr/local/share/ca-certificates/sensu-ca.crt
sudo update-ca-certificates
```

SHELL

```
chmod 644 /etc/sensu/tls/ca.pem
chown root /etc/sensu/tls/ca.pem
sudo yum install -y ca-certificates
sudo update-ca-trust force-enable
sudo ln -s /etc/sensu/tls/ca.pem /etc/pki/ca-trust/source/anchors/sensu-ca.pem
sudo update-ca-trust
```

SHELL

```
Import the root CA certificate on the Mac.
Double-click the root CA certificate to open it in Keychain Access.
The root CA certificate appears in login.
Copy the root CA certificate to System to ensure that it is trusted by all users and
local system processes.
Open the root CA certificate, expand Trust, select Use System Defaults, and save your
changes.
Reopen the root CA certificate, expand Trust, select Always Trust, and save your
changes.
Delete the root CA certificate from login.
```

SHELL

```
Press Windows+R to open the Run dialog.
Type "MMC" (without quotation marks) in the Run dialog and press Enter to open the
MMC console.
In the MMC console, expand the Certificates (Local Computer) node and navigate to
Trusted Root Certification Authorities > Certificates.
Right-click the Trusted Root Certification Authorities > Certificates folder and
select All Tasks > Import to open the Certificate Import dialog.
In the Certificate Import dialog, click Next and browse to the location where the
root CA certificate is stored.
Select the root CA certificate file and click Open.
Click Next, click Next, and click Finish.
```

Renew self-generated certificates

To keep your Sensu deployment running smoothly, renew your self-generated certificates before they expire.Depending on how your certificates are configured, one backend certificate may expire before the others or all three backend certificates may expire at the same time.The agent certificate also expires.

This section explains how to find certificate expiration dates, confirm whether certificates have already expired, and renew certificates.

Find certificate expiration dates

Use this check to find certificate expiration dates so you can renew certificates before they expire and

avoid observability interruptions.

Before you run the check, replace `<cert-name>.pem` in the command with the name of the certificate you want to check (for example, `backend-1.example.com.pem`).

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: expired_certs
spec:
  command: openssl x509 -noout -enddate -in <cert-name>.pem
  subscriptions:
  - system
  publish: true
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "expired_certs"
  },
  "spec": {
    "command": "openssl x509 -noout -enddate -in <cert-name>.pem",
    "subscriptions": [
      "system"
    ],
    "publish": true
  }
}
```

The check output will be in the format `notAfter=Month Day HH:MM:SS Year Timezone`.For example:

```
notAfter=Jul  3 22:23:50 2021 GMT
```

Add a handler to send the check output as a notification or to a log file.

Identify expired certificates

The following `sensuctl cluster health` response indicates that one backend certificate is expired:

```
Error: GET "/health": Get https://localhost:8080/health?timeout=3: x509: certificate has expired or is not yet valid
```

The log for the expired backend will be similar to this example:

```
backend-1.example.com | {"component":"etcd","level":"warning","msg":"health check for peer a95ca1cdb0b1fcc3 could not connect: remote error: tls: bad certificate (prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-22T20:40:54Z"}
backend-1.example.com | {"component":"etcd","level":"warning","msg":"health check for peer a95ca1cdb0b1fcc3 could not connect: remote error: tls: bad certificate (prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-22T20:40:54Z"}
```

If you restart the cluster with one expired backend certificate, the `sensuctl cluster health` response will include an error:

```
Error: GET "/health": failed to request new refresh token; client returned 'Post https://localhost:8080/auth/token: EOF'
```

When all three backend certificates are expired, the log will be similar to this example:

```
backend-1.example.com | {"component":"etcd","level":"warning","msg":"health check for peer a95ca1cdb0b1fcc3 could not connect: x509: certificate has expired or is not yet valid (prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-25T17:49:53Z"}
backend-2.example.com | {"component":"etcd","level":"warning","msg":"health check for peer 4cc36e198efb22e8 could not connect: x509: certificate has expired or is not
```

```
yet valid (prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-25T17:49:16Z"}
backend-3.example.com | {"component":"etcd","level":"warning","msg":"health check for peer 8425a7b2d2ee8597 could not connect: x509: certificate has expired or is not yet valid (prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-25T17:49:16Z"}
```

If you restart the cluster with three expired backend certificates, the `sensuctl cluster health` response will include an error:

```
Error: GET "/health": Get https://127.0.0.1:8080/health?timeout=3: EOF
```

The following `sensuctl cluster health` response helps confirm that all three backend certificates are expired, together with the log warning and restart error examples:

```
=== Etcd Cluster ID: 45c04eab9efc0d11
      ID           Name                    Error          Healthy
 ——————————————— ———————————————————— ———————————————————————————— —————————
  a95ca1cdb0b1fcc3  backend-1.example.com   context deadline exceeded   false
  8425a7b2d2ee8597  backend-2.example.com   context deadline exceeded   false
  4cc36e198efb22e8  backend-3.example.com   context deadline exceeded   false
```

An expired agent certificate does not cause any errors or log messages to indicate the expiration. Use the certificate expiration check to find the agent certificate expiration date.

Renew certificates

To renew your certificates, whether they expired or not, follow the steps to create a CA, generate backend certificates, or generate an agent certificate. The new certificate will override the existing certificate.

After you save the new certificates, restart each backend:

```
sudo systemctl start sensu-backend
```

Next step: Secure Sensu

Now that you have generated the required certificates, follow Secure Sensu to make your Sensu installation production-ready.

Secure Sensu

As with any piece of software, it is critical to minimize any attack surface the software exposes. Sensu is no different.

This reference describes the components you need to secure to make Sensu production-ready, including etcd peer communication, the Sensu API and web UI, and Sensu agent-to-server communication. It also describes agent mutual transport layer security (mTLS) authentication, which is required for secrets management.

Before you can secure Sensu, you must generate the certificates you will need. After you generate certificates, follow this reference to secure Sensu for production.

***NOTE**: As of Go 1.15, certificates must include their Common Name (CN) as a Subject Alternative Name (SAN) field. To prevent connection errors, follow Generate certificates to make sure your certificates' SAN fields include their CNs.*

Secure etcd peer communication

***WARNING**: You must update the default configuration for Sensu's embedded etcd with an explicit, non-default configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

To properly secure etcd communication, replace the default configuration option values in your backend store configuration in `/etc/sensu/backend.yml` as follows:

1. Replace the placeholder with the path to your certificate and key for the `etcd-cert-file` and `etcd-key-file` to secure client communication:

```
etcd-cert-file: "/etc/sensu/tls/backend-1.example.com.pem"
etcd-key-file: "/etc/sensu/tls/backend-1.example.com-key.pem"
```

2. Replace the placeholder with the path to your certificate and key for the `etcd-peer-cert-file` and `etcd-peer-key-file` to secure cluster communication:

```
etcd-peer-cert-file: "/etc/sensu/tls/backend-1.example.com.pem"
etcd-peer-key-file: "/etc/sensu/tls/backend-1.example.com-key.pem"
```

3. Replace the placeholder with the path to your `ca.pem` certificate for the `etcd-trusted-ca-file` and `etcd-peer-trusted-ca-file` to secure communication with the etcd client server and between etcd cluster members:

```
etcd-trusted-ca-file: "/etc/sensu/tls/ca.pem"
etcd-peer-trusted-ca-file: "/etc/sensu/tls/ca.pem"
```

4. Add non-default values for `etcd-listen-client-urls`, `etcd-listen-peer-urls`, and `etcd-initial-advertise-peer-urls`:

```
etcd-listen-client-urls: "https://localhost:2379"
etcd-listen-peer-urls: "https://localhost:2380"
etcd-advertise-client-urls: "https://localhost:2379"
etcd-initial-advertise-peer-urls: "https://localhost:2380"
```

***NOTE**: If you are securing a cluster, use your backend node IP address instead of `localhost` in the non-default values for `etcd-listen-client-urls`, `etcd-listen-peer-urls`, and `etcd-initial-advertise-peer-urls`.*

5. Set `etcd-client-cert-auth` and `etcd-peer-client-cert-auth` to `true` to ensure that etcd only allows connections from clients and peers that present a valid, trusted certificate:

```
etcd-client-cert-auth: "true"
etcd-peer-client-cert-auth: "true"
```

Because etcd does not require authentication by default, you must set the `etcd-client-cert-auth` and `etcd-peer-client-cert-auth` configuration options to `true` to secure Sensu's

embedded etcd datastore against unauthorized access.

> ***NOTE**: The Sensu backend reference includes more information about each etcd store configuration option.*

Secure the Sensu agent API, HTTP API, and web UI

The Sensu Go agent API, HTTP API, and web UI use a common stanza in `/etc/sensu/backend.yml` to provide the certificate, key, and CA file needed to provide secure communication.

> ***NOTE**: By changing these configuration options, the server will communicate using transport layer security (TLS) and expect agents that connect to it to use the WebSocket secure protocol. For communication to continue, you must complete the configuration in this section **and** in the Sensu agent-to-server communication section.*

Configure the following backend secure sockets layer (SSL) attributes in `/etc/sensu/backend.yml`:

1. Replace the placeholders with the paths to your CA root, backend certificate, and backend key files for the `trusted-ca-file`, `cert-file`, and `key-file` configuration options:

```
trusted-ca-file: "/etc/sensu/tls/ca.pem"
cert-file: "/etc/sensu/tls/backend-1.example.com.pem"
key-file: "/etc/sensu/tls/backend-1.example.com-key.pem"
```

2. Set the `insecure-skip-tls-verify` configuration option to `false`:

```
insecure-skip-tls-verify: false
```

3. When you provide these cert-file and key-file configuration options, the agent WebSocket API and HTTP API will serve requests over SSL/TLS (https). For this reason, you must also specify `https://` schema for the `api-url` configuration option for backend API configuration:

```
  api-url: "https://localhost:8080"
```

Restart the `sensu-backend` service:

```
sudo systemctl restart sensu-backend
```

After you restart the `sensu-backend` service, the configuration options will load and you will able to access the web UI at https://localhost:3000. Configuring these options will also ensure that agents can communicate securely.

***NOTE**: The Sensu backend reference includes more information about each API and web UI security configuration option.*

Specify a separate web UI certificate and key

You can use the same certificates and keys to secure etcd, the HTTP API, and the web UI. However, if you prefer, you can use a separate certificate and key for the web UI (for example, a commercially purchased certificate and key).

To do this, add the `dashboard-cert-file` and `dashboard-key-file` configuration options for backend SSL configuration in `/etc/sensu/backend.yml`:

```
dashboard-cert-file: "/etc/sensu/tls/separate-webui-cert.pem"
dashboard-key-file: "/etc/sensu/tls/separate-webui-key.pem"
```

***NOTE**: If you do not specify a separate certificate and key for the web UI with `dashboard-cert-file` and `dashboard-key-file`, Sensu uses the certificate and key specified for the `cert-file` and `key-file` configuration options for the web UI. The Sensu backend reference includes more information about the `dashboard-cert-file` and `dashboard-key-file` web UI configuration options.*

Secure Sensu agent-to-server communication

***NOTE**: If you change the agent configuration to communicate via WebSocket Secure protocol, the agent will no longer communicate over a plaintext connection. For communication to continue, you must complete the steps in this section **and** secure the Sensu agent API, HTTP API, and web UI.*

By default, an agent uses the insecure `ws://` transport. Here's an example for agent configuration in `/etc/sensu/agent.yml`:

```
backend-url:
  - "ws://127.0.0.1:8081"
```

To use WebSocket over SSL/TLS (wss), change the `backend-url` value to the `wss://` schema in `/etc/sensu/agent.yml`:

```
backend-url:
  - "wss://127.0.0.1:8081"
```

The agent will connect to Sensu backends over wss. Remember, if you change the configuration to wss, plaintext communication will not be possible.

You can also provide a trusted CA root certificate file as part of the agent configuration (named `ca.pem` in the example in Generate certificates). If you will start the agent via `sensu-agent start`, pass the `--trusted-ca-file` flag with the start command. Otherwise, include the `trusted-ca-file` configuration option in the agent configuration in `/etc/sensu/agent.yml`:

```
trusted-ca-file: "/etc/sensu/tls/ca.pem"
```

***NOTE**: If you are creating a Sensu cluster, every cluster member needs to be present in the configuration. Read Run a Sensu cluster for more information about how to configure agents for a clustered configuration.*

Optional: Configure Sensu agent mTLS authentication

***COMMERCIAL FEATURE**: Access client mutual transport layer security (mTLS) authentication in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

By default, Sensu agents require username and password authentication to communicate with Sensu backends. For Sensu's default user credentials and details about configuring Sensu role-based access control (RBAC), read the RBAC reference.

Alternatively, Sensu agents can use mTLS for authenticating to the backend WebSocket transport. When agent mTLS authentication is enabled, agents do not need to send password credentials to backends when they connect. To use secrets management, Sensu agents must be secured with mTLS. In addition, when using mTLS authentication, agents do not require an explicit user in Sensu. Sensu agents default to authenticating as the `agent` user and using permissions granted to the `system:agents` group by the `system:agents` cluster role and cluster role binding.

You can still bind agents to a specific user when the `system:agents` group is problematic. For this use case, create a user that matches the Common Name (CN) of the agent's certificate.

***NOTE**: Sensu agents need to be able to create events in the agent's namespace. To ensure that agents with incorrect CN fields can't access the backend, remove the default `system:agents` group.*

For example, if you have a certificate named `client.pem`, you can run the following command to view the certificate's CN with openssl:

```
openssl x509 -in client.pem -text -noout
```

The response should be similar to this example:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            37:57:7b:04:1d:67:63:7b:ff:ae:39:19:5b:55:57:80:41:3c:ec:ff
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = CA
        Validity
            Not Before: Sep 26 18:58:00 2019 GMT
```

```
            Not After : Sep 24 18:58:00 2024 GMT
        Subject: CN = client
...
```

The `Subject:` field indicates the certificate's CN is `client`, so to bind the agent to a particular user in Sensu, create a user called `client`.

To enable agent mTLS authentication:

1. Create and distribute a new Certificate Authority (CA) root certificate and new agent and backend certificates and keys according to the Generate certificates guide.

2. Add the following configuration options and values to the backend configuration `/etc/sensu/backend.yml`:

   ```
   agent-auth-cert-file: "/etc/sensu/tls/backend-1.example.com.pem"
   agent-auth-key-file: "/etc/sensu/tls/backend-1.example.com-key.pem"
   agent-auth-trusted-ca-file: "/etc/sensu/tls/ca.pem"
   ```

3. Add the following configuration options and values to the agent configuration in `/etc/sensu/agent.yml`:

   ```
   cert-file: "/etc/sensu/tls/agent.pem"
   key-file: "/etc/sensu/tls/agent-key.pem"
   trusted-ca-file: "/etc/sensu/tls/ca.pem"
   ```

You can use use certificates for authentication that are distinct from other communication channels used by Sensu, like etcd or the API.However, deployments can also use the same certificates and keys for etcd peer and client communication, the HTTP API, and agent authentication without issues.

Certificate revocation check

The Sensu backend checks certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP) endpoints for agent mTLS, etcd client, and etcd peer connections whose remote sides present X.509 certificates that provide CRL and OCSP revocation information.

Optional: Configure Sensu for FIPS compliance

Sensu provides a Linux amd64 OpenSSL-linked build that supports the Federal Information Processing Standard (FIPS) for Federal Risk and Authorization Management Program (FedRAMP) compliance.

The Sensu build with FIPS-mode configuration options is linked with the FIPS 140-2 validated cryptographic library. Sensu builds comply with the FIPS-mode kernel option to enforce FIPS systemwide in Red Hat Enterprise Linux (RHEL). Contact Sensu to request the build with FIPS support.

Sensu backends and agents will work on systems with FIPS kernel mode if the `require-fips` and `require-openssl` configuration options are set to `true` in the backend and agent configuration files Sensu backends and agents that have `require-fips` enabled will *not* work on systems without FIPS kernel mode.

Sensu backends on systems with FIPS kernel mode will work with PostgreSQL on systems with FIPS kernel mode. For PostgreSQL on systems *without* FIPS kernel mode, Sensu backends with FIPS kernel mode will work as long as the PostgreSQL system supports FIPS-compliant ciphers/cipher suites.

Sensu agents and sensuctl on systems with and without FIPS kernel mode can connect to Sensu backends on systems with FIPS kernel mode.

Configuration example for embedded etcd

To configure the Sensu backend for FIPS mode with embedded etcd, update the backend configuration file at `/etc/sensu/backend.yml` to use the following settings:

```
# fips configuration
require-openssl: true
require-fips: true

# etcd configuration
etcd-listen-client-urls: "https://localhost:2379"
etcd-listen-peer-urls: "https://localhost:2380"
etcd-advertise-client-urls: "https://localhost:2379"
etcd-initial-advertise-peer-urls: "https://localhost:2380"

# etcd client tls configuration
etcd-client-cert-auth: "true"
```

```
etcd-trusted-ca-file: "/etc/sensu/tls/ca.pem"
etcd-cert-file: "/etc/sensu/tls/centos-7-fips-1-backend.pem"
etcd-key-file: "/etc/sensu/tls/centos-7-fips-1-backend-key.pem"

# etcd peer tls configuration
etcd-peer-client-cert-auth: "true"
etcd-peer-trusted-ca-file: "/etc/sensu/tls/ca.pem"
etcd-peer-cert-file: "/etc/sensu/tls/centos-7-fips-1-backend.pem"
etcd-peer-key-file: "/etc/sensu/tls/centos-7-fips-1-backend-key.pem"

# api configuration
api-url: "https://localhost:8080"

# api tls configuration
insecure-skip-tls-verify: false
trusted-ca-file: "/etc/sensu/tls/ca.pem"
cert-file: "/etc/sensu/tls/centos-7-fips-1-backend.pem"
key-file: "/etc/sensu/tls/centos-7-fips-1-backend-key.pem"
```

***NOTE**: If you are securing a cluster, use your backend node IP address instead of `localhost`.*

Configuration example for external etcd

To configure the Sensu backend for FIPS mode with external etcd, update the backend configuration file at `/etc/sensu/backend.yml` to use the following settings:

```
# fips configuration
require-openssl: true
require-fips: true

# etcd configuration
etcd-trusted-ca-file: "/etc/sensu/tls/ca.pem"
etcd-cert-file: "/etc/sensu/tls/centos-7-fips-1-backend.pem"
etcd-key-file: "/etc/sensu/tls/centos-7-fips-1-backend-key.pem"
etcd-client-urls: "https://localhost:2379"
no-embed-etcd: true

# api configuration
```

```
api-url: "https://localhost:8080"

# api tls configuration
insecure-skip-tls-verify: false
trusted-ca-file: "/etc/sensu/tls/ca.pem"
cert-file: "/etc/sensu/tls/centos-7-fips-1-backend.pem"
key-file: "/etc/sensu/tls/centos-7-fips-1-backend-key.pem"
```

Use the following settings in your etcd configuration:

```
name: "centos-7-fips-1"
data-dir: "/var/lib/etcd-external"
auto-compaction-mode: "revision"
auto-compaction-retention: "2"

# cluster config
initial-cluster-token: "sup3rs3cr3t"
initial-cluster: "centos-7-fips-1=https://centos-7-fips-1:2380"
initial-cluster-state: "new"

# etcd configuration
listen-client-urls: "https://localhost:2379"
listen-peer-urls: "https://localhost:2380"
advertise-client-urls: "https://localhost:2379"
initial-advertise-peer-urls: "https://localhost:2380"

# etcd client tls configuration
client-transport-security:
  client-cert-auth: true
  trusted-ca-file: /etc/etcd/tls/ca.pem
  cert-file: /etc/etcd/tls/centos-7-fips-1-backend.pem
  key-file: /etc/etcd/tls/centos-7-fips-1-backend-key.pem
  auto-tls: false

# etcd peer tls configuration
peer-transport-security:
  client-cert-auth: true
  trusted-ca-file: /etc/etcd/tls/ca.pem
  cert-file: /etc/etcd/tls/centos-7-fips-1-backend.pem
  key-file: /etc/etcd/tls/centos-7-fips-1-backend-key.pem
```

```
    auto-tls: false
```

***NOTE**: If you are securing a cluster, use your backend node IP address instead of `localhost`.*

Next step: Run a Sensu cluster

Well done! Your Sensu installation should now be secured with TLS. The last step before you deploy Sensu is to set up a Sensu cluster.

Secure PostgreSQL

This guide describes how to secure communication between Sensu and the PostgreSQL event store using certificate authentication. When deploying Sensu for use outside of a local development environment, you should secure it using transport layer security (TLS).

To learn how to secure communications between Sensu and its agents, read Generate certificates for your Sensu installation and Secure Sensu.

> ***NOTE**: This guide describes one option for securing communication between Sensu and PostgreSQL and is intended as a starting point. Your organization's needs may require a different approach.*

Prerequisites

To use this guide, you must have:

- A running Sensu deployment.
- A running PostgreSQL instance that you've configured according to Scale Sensu Go with Enterprise datastore. The commands in this guide use PostgreSQL version 14.

Install cfssl

The CloudFlare cfssl toolkit is released as a collection of command-line tools.

If you followed Generate certificates for your Sensu installation, you already downloaded and installed the Cloudflare cfssl toolkit. If not, run the following commands:

SHELL

```
sudo curl -s -L -o /bin/cfssl
https://github.com/cloudflare/cfssl/releases/download/v1.6.2/cfssl_1.6.2_linux_amd64
sudo curl -s -L -o /bin/cfssljson
https://github.com/cloudflare/cfssl/releases/download/v1.6.2/cfssljson_1.6.2_linux_am
```

```
d64
sudo curl -s -L -o /bin/cfssl-certinfo
https://github.com/cloudflare/cfssl/releases/download/v1.6.2/cfssl-
certinfo_1.6.2_linux_amd64
sudo chmod +x /bin/cfssl*
```

SHELL

```
# Update apt repos
sudo apt-get update

# Install cfssl
sudo apt-get install golang-cfssl
```

To verify that cfssl is installed, run:

```
cfssl version
```

Create a Certificate Authority (CA)

Follow these steps to create a CA with cfssl and cfssljson:

1. Create `/etc/sensu/tls` (which does not exist by default):

   ```
   mkdir -p /etc/sensu/tls
   ```

2. Navigate to the new `/etc/sensu/tls` directory:

   ```
   cd /etc/sensu/tls
   ```

3. Create the CA:

```
echo '{"CN":"Sensu Test CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert
-initca - | cfssljson -bare ca -
```

4. Define signing parameters and profiles:

```
echo '{"signing":{"default":{"expiry":"17520h","usages":["signing","key
encipherment","client auth"]},"profiles":{"postgresql":{"usages":
["signing","key encipherment","server auth","client
auth"],"expiry":"4320h"},"backend":{"usages":["signing","key
encipherment","client auth"],"expiry":"4320h"}}}}' > ca-config.json
```

> ***NOTE**: We suggest a 6-month expiry duration for security, but you can use any duration you prefer when you define the `expiry` attribute value in the signing parameters.*

You should now have a directory at `/etc/sensu/tls` that contains the following files:

filename	description
`ca.pem`	CA root certificate. Required for all systems running the Sensu backend or agent. The agent and backend use `ca.pem` to validate server certificates at connection time.
`ca-key.pem`	CA root certificate private key.
`ca-config.json`	CA signing parameters and profiles. Not used by Sensu.
`ca.csr`	Certificate signing request for the CA root certificate. Not used by Sensu.

Generate certificate and key for PostgreSQL

Next, generate the certificates you need for PostgreSQL.

This guide assumes your PostgreSQL instance is reachable via a `10.0.0.x` IP address, a fully qualified name (for example, `postgres.example.com`), and an unqualified name (for example, `postgres`):

Unqualified name	IP address	Fully qualified domain name (FQDN)	Additional names
postgres	10.0.0.43	postgres.example.com	localhost, 127.0.0.1

The additional names for localhost and 127.0.0.1 are added here for convenience and are not strictly required.

- The values provided for the ADDRESS variable will be used to populate the certificate's SAN records. For systems with multiple hostnames and IP addresses, add each to the comma-delimited value of the ADDRESS variable.
- The value provided for the NAME variable will be used to populate the certificate's CN record. It will also be used in the names for the `*.pem` and `*.csr` files.

For example:

```
export ADDRESS=localhost,127.0.0.1,10.0.0.43,postgres,postgres.example.com
export NAME=postgres.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="postgresql" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

The `/etc/sensu/tls` directory should now include the following files for your PostgreSQL instance:

filename	description
`postgres.example.com.pem`	The certificate that your PostgreSQL instance will use.
`postgres.example.com-key.pem`	The private key that your PostgreSQL instance will use.
`postgres.example.com.csr`	Certificate signing request for the PostgreSQL certificate. Not used.

Generate certificate and key for your Sensu backend

Just like the certificate and key for PostgreSQL, you'll need a certificate and key for the Sensu backend.

To generate the backend certificate and key, run:

```
export POSTGRES_USERNAME=sensu
echo '{"CN":"'$POSTGRES_USERNAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' |
cfssl gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="" -
profile=backend - | cfssljson -bare $POSTGRES_USERNAME
```

You'll also need to change the ownership of the certificate files to the `sensu` user:

```
chown -R sensu:sensu /etc/sensu/tls
```

You should now have the following files in your `/etc/sensu/tls` directory, which the Sensu backend will use to communicate with PostgreSQL:

filename	description
`sensu.pem`	The certificate that your Sensu backend will use.
`sensu-key.pem`	The private key that your Sensu backend will use.
`sensu.csr`	Certificate signing request for the Sensu backend certificate. Not used.

Now that you have the required certificates and keys, you can configure Sensu to use certificate authentication with PostgreSQL.

***WARNING**: Once you've generated all of your certificates, delete the `ca-key.pem` file from the `/etc/sensu/tls` directory. The `ca-key.pem` file contains sensitive information and is only needed on your PostgreSQL instance.*

Configure Sensu to use certificate authentication with PostgreSQL

***NOTE**: The Sensu backend uses the libpq library to make connections to PostgreSQL. This library supports a number of environment variables that can be injected into the PostgreSQL data source name (DSN) and are loaded at runtime using the system's environment variable file. These environment variables allow you to customize the Sensu backend's PostgreSQL DSN construction to suit your needs.*

Working from your Sensu backend, follow these steps to configure Sensu to use certificate authentication with PostgreSQL:

1. Define the environment variables that tell the Sensu backend to use a certificate to authenticate to PostgreSQL:

   SHELL

   ```
   echo 'PGUSER=sensu
   PGSSLMODE="verify-full"
   PGSSLCERT="/etc/sensu/tls/sensu.pem"
   PGSSLKEY="/etc/sensu/tls/sensu-key.pem"
   PGSSLROOTCERT="/etc/sensu/tls/ca.pem"' | sudo tee /etc/sysconfig/sensu-backend
   ```

   SHELL

   ```
   echo 'PGUSER=sensu
   PGSSLMODE="verify-full"
   PGSSLCERT="/etc/sensu/tls/sensu.pem"
   PGSSLKEY="/etc/sensu/tls/sensu-key.pem"
   PGSSLROOTCERT="/etc/sensu/tls/ca.pem"' | sudo tee /etc/default/sensu-backend
   ```

   Do not restart your backend to load the environment variables yet.

2. Adjust the Sensu datastore connection with sensuctl:

   ```
   echo 'type: PostgresConfig
   api_version: store/v1
   ```

```
metadata:
  name: sensu_postgres
spec:
  dsn: "postgresql://sensu:mypass@postgres.example.com:5432/sensu_events"
  pool_size: 20
  strict: false' | sudo tee postgresconfig.yml

sensuctl create -f postgresconfig.yml
```

***NOTE**: Setting `strict: false` in the configuration helps ensure that the Sensu backend will remain active and able to process events even in case of a configuration mistake.*

3. Confirm that the connection to your PostgreSQL instance is healthy:

```
curl http://localhost:8080/health
```

The response should be similar to this example, with `true` values for both `Active` and `Healthy`:


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 13217573501179607000,
      "MemberIDHex": "b76e4111d26d35e2",
      "Name": "sensu.example.com",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 11959078708079102000,
    "member_id": 6370351775894128000,
    "raft_term": 4242
  },
  "PostgresHealth": [
    {
```

```
      "Name": "sensu_postgres",
      "Active": true,
      "Healthy": true
    }
  ]
}
```

Now that you've confirmed that the Sensu backend can connect to your PostgreSQL instance, you can configure PostgreSQL to use TLS.

Configure PostgreSQL to use TLS

To configure your PostgreSQL instance to use TLS:

1. Copy your PostgreSQL certificate files from your Sensu backend. From the `/etc/sensu/tls` directory, run:

```
scp postgres.example.com* postgres.example.com:/home/user
scp ca.pem postgres.example.com:/home/user
```

2. From your PostgreSQL instance, create a new directory and move your PostgreSQL certificate files from your Sensu backend:

SHELL

```
sudo mkdir /var/lib/pgsql/14/data/tls
cd /var/lib/pgsql/14/data/tls
cp /home/user/postgres.example.com* /var/lib/pgsql/14/data/tls/
cp /home/user/ca.pem /var/lib/pgsql/14/data/tls/
chown -R postgres:postgres /var/lib/pgsql/14/data
```

SHELL

```
sudo mkdir /etc/postgresql/14/main/tls
cd /etc/postgresql/14/main/tls
cp /home/user/postgres.example.com* /etc/postgresql/14/main/tls/
cp /home/user/ca.pem /etc/postgresql/14/main/tls/
chown -R postgres:postgres /etc/postgresql/14/main/
```

3. Open the PostgreSQL configuration file `postgresql.conf` in your code editor and edit the following lines to enable TLS:

SHELL

```
# vim /var/lib/pgsql/14/data/postgresql.conf

# - SSL -

ssl = on
ssl_ca_file = '/var/lib/pgsql/14/data/tls/ca.pem'
ssl_cert_file = '/var/lib/pgsql/14/data/tls/postgres.example.com.pem'
ssl_key_file = '/var/lib/pgsql/14/data/tls/postgres.example.com-key.pem'
```

SHELL

```
# vim /etc/postgresql/14/main/postgresql.conf

# - SSL -

ssl = on
ssl_ca_file = '/etc/postgresql/14/main/tls/ca.pem'
ssl_cert_file = '/etc/postgresql/14/main/tls/postgres.example.com.pem'
ssl_key_file = '/etc/postgresql/14/main/tls/postgres.example.com-key.pem'
```

Save your changes and close the file.

4. Open the `pg_hba.conf` file in your Linux distribution and add the following lines to configure host-based authentication to accept certificates only when accessing the `sensu_events` database:

SHELL

```
# /var/lib/pgsql/14/data/pg_hba.conf (file location)

# Prevent "postgres" superuser login via a certificate
hostssl all             postgres        ::/0                    reject
hostssl all             postgres        0.0.0.0/0               reject
```

```
# Rules for Sensu DB connection
hostssl sensu_events    sensu           0.0.0.0/0               cert
```

SHELL

```
# /etc/postgresql/14/main/pg_hba.conf (file location)

# Prevent "postgres" superuser login via a certificate
hostssl all             postgres        ::/0                    reject
hostssl all             postgres        0.0.0.0/0               reject

# Rules for Sensu DB connection
hostssl sensu_events    sensu           0.0.0.0/0               cert
```

Take care to add the new lines in the positions shown in the following example:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer

# Prevent "postgres" superuser login via a certificate
hostssl all             postgres        ::/0                    reject
hostssl all             postgres        0.0.0.0/0               reject

# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256

# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256

# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
hostssl sensu_events    sensu           0.0.0.0/0               cert
```

5. Restart PostgreSQL:

SHELL

```
sudo systemctl restart postgresql-14.service
```

SHELL

```
sudo systemctl restart postgresql.service
```

Now that you've configured PostgreSQL to use TLS and your Sensu user is required to authenticate with a certificate, complete one final step to ensure that the Sensu backend uses the environment variables set earlier in this guide when constructing the PostgreSQL DSN.

Validate Sensu backend configuration for PostgreSQL

After restarting PostgreSQL, the Sensu user should ***not*** be able to communicate with PostgreSQL because it requires certificate authentication for the `sensu_events` database.Run:

```
curl http://localhost:8080/health
```

The response should include `false` values for PostgresHealth.Active and PostgresHealth.Healthy:


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 13217573501179607000,
      "MemberIDHex": "b76e4111d26d35e2",
      "Name": "sensu.example.com",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 11959078708079102000,
    "member_id": 6370351775894128000,
    "raft_term": 4242
  },
  "PostgresHealth": [
    {
```

```
      "Name": "sensu_postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```

For Sensu to use certificate authentication, you must restart the backend service to load the environment variables set previously:

```
sudo systemctl restart sensu-backend.service
```

To validate that your Sensu backend can reach PostgreSQL and authenticate after restarting, run the following command:

```
curl http://localhost:8080/health
```

The response should be similar to the following example. If the `Active` and `Healthy` attributes are not **both** `true`, stop and troubleshoot your connection to PostgreSQL *before* you continue:

```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 13217573501179607000,
      "MemberIDHex": "b76e4111d26d35e2",
      "Name": "sensu.example.com",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 11959078708079102000,
    "member_id": 6370351775894128000,
    "raft_term": 4242
  },
  "PostgresHealth": [
```

```
    {
      "Name": "sensu_postgres",
      "Active": true,
      "Healthy": true
    }
  ]
}
```

Optional step: Require PostgreSQL as event store

To force Sensu to always use PostgreSQL as the event store instead of falling back to etcd if PostgreSQL becomes unavailable, set `strict: true` in your PostgreSQL configuration file.

If you prefer to use etcd as a fallback, skip this step. Using etcd as a fallback may result in disk quota alarms and etcd unavailability, especially in environments with a large number of events.

To set `strict: true` in your PostgreSQL configuration file, run:

```
echo 'type: PostgresConfig
api_version: store/v1
metadata:
  name: sensu_postgres
spec:
  dsn: "postgresql://postgres.example.com:5432/sensu_events"
  pool_size: 20
  strict: true' | sudo tee postgresconfig.yml

sensuctl create -f postgresconfig.yml
```

Your backend will now use PostgreSQL exclusively for storing events.

To view your PostgresConfig definition and confirm that it is updated, run:

```
sensuctl dump store/v1.PostgresConfig --format yaml
```

Run a Sensu cluster

To deploy Sensu for use outside of a local development environment, first decide whether you want to run a Sensu cluster.

A Sensu cluster is a group of at least three sensu-backend nodes, each connected to a shared database provided either by Sensu's embedded etcd or an external etcd cluster. Creating a Sensu cluster ultimately configures an etcd cluster.

Clustering improves Sensu's availability, reliability, and durability. It allows you to absorb the loss of a backend node, prevent data loss, and distribute the network load of agents. If you have a healthy clustered backend, you only need to make Sensu API calls to any one of the cluster members. The cluster protocol will replicate your changes to all cluster members.

Scaling a single backend to a cluster or migrating a cluster from cleartext HTTP to encrypted HTTPS without downtime can require a number of tedious steps. For this reason, we recommend that you **decide whether your deployment will require clustering as part of your initial planning effort**.

No matter whether you deploy a single backend or a clustered configuration, begin by securing Sensu with transport layer security (TLS). The first step in setting up TLS is to generate the certificates you need. Then, follow our Secure Sensu guide to make Sensu production-ready.

After you've secured Sensu, continue reading this document to set up and update a clustered configuration.

> ***NOTE**: We recommend using a load balancer to evenly distribute agent connections across a cluster.*

Configure a cluster

The sensu-backend arguments for its store mirror the etcd configuration flags, but the Sensu configuration options are prefixed with `etcd`. For more detailed descriptions of the different arguments read the etcd documentation or Sensu backend reference.

You can configure a Sensu cluster in a couple different ways — we'll show you a few below — but you should adhere to some etcd cluster guidelines as well:

> The recommended etcd cluster size is 3, 5 or 7, which is decided by the fault tolerance requirement. A 7-member cluster can provide enough fault tolerance in most cases. While a larger cluster provides better fault tolerance, the write performance reduces since data needs to be replicated to more machines. It is recommended to have an odd number of members in a cluster. Having an odd cluster size doesn't change the number needed for majority, but you gain a higher tolerance for failure by adding the extra member. *etcd2 Admin Guide*

We also recommend using stable platforms to support your etcd instances (review etcd's supported platforms).

***NOTE**: If a cluster member is started before it is configured to join a cluster, the member will persist its prior configuration to disk. For this reason, you must remove any previously started member's etcd data by stopping sensu-backend and deleting the contents of `/var/lib/sensu/sensu-backend/etcd` before proceeding with cluster configuration.*

Docker

If you prefer to stand up your Sensu cluster within Docker containers, check out the Sensu Go Docker configuration. This configuration defines three sensu-backend containers and three sensu-agent containers.

Traditional computer instance

***NOTE**: The remainder of this guide describes on-disk configuration. If you are using an ephemeral computer instance, you can use `sensu-backend start --help` to list etcd command line flags. The configuration file entries in the rest of this guide translate to `sensu-backend` flags.*

Sensu backend configuration

***WARNING**: You must update the default configuration for Sensu's embedded etcd with an explicit, non-default configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

The examples in this section are configuration snippets from `/etc/sensu/backend.yml` using a three-node cluster. The nodes are named `backend-1.example.com`, `backend-2.example.com` and `backend-3.example.com` with IP addresses `10.0.0.1`, `10.0.0.2` and `10.0.0.3`, respectively.

> ***NOTE**: This backend configuration assumes you have set up and installed the sensu-backend on all the nodes used in your cluster. Follow the Install Sensu guide if you have not already done this.*

Store configuration for backend-1.example.com/10.0.0.1

```
etcd-advertise-client-urls: "https://10.0.0.1:2379"
etcd-listen-client-urls: "https://10.0.0.1:2379"
etcd-listen-peer-urls: "https://0.0.0.0:2380"
etcd-initial-cluster: "backend-1.example.com=https://10.0.0.1:2380,backend-
2.example.com=https://10.0.0.2:2380,backend-3.example.com=https://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "https://10.0.0.1:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: "unique_token_for_this_cluster"
etcd-name: "backend-1.example.com"
```

Store configuration for backend-2.example.com/10.0.0.2

```
etcd-advertise-client-urls: "https://10.0.0.2:2379"
etcd-listen-client-urls: "https://10.0.0.2:2379"
etcd-listen-peer-urls: "https://0.0.0.0:2380"
etcd-initial-cluster: "backend-1.example.com=https://10.0.0.1:2380,backend-
2.example.com=https://10.0.0.2:2380,backend-3.example.com=https://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "https://10.0.0.2:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: "unique_token_for_this_cluster"
etcd-name: "backend-2.example.com"
```

Store configuration for backend-3.example.com/10.0.0.3

```
etcd-advertise-client-urls: "https://10.0.0.3:2379"
etcd-listen-client-urls: "https://10.0.0.3:2379"
etcd-listen-peer-urls: "https://0.0.0.0:2380"
```

```
etcd-initial-cluster: "backend-1.example.com=https://10.0.0.1:2380,backend-
2.example.com=https://10.0.0.2:2380,backend-3.example.com=https://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "https://10.0.0.3:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: "unique_token_for_this_cluster"
etcd-name: "backend-3.example.com"
```

***WARNING**: To properly secure etcd communication, replace the default URLs for `etcd-advertise-client-urls`, `etcd-listen-client-urls`, `etcd-listen-peer-urls`, and `etcd-initial-cluster` in the store configurations for your backends with non-default values.*

Specify the same `etcd-initial-cluster-token` value for all three backends. This allows etcd to generate unique cluster IDs and member IDs even for clusters that have otherwise identical configurations and prevents cross-cluster-interaction.

After you configure each node as described in these examples, start each sensu-backend:

```
sudo systemctl start sensu-backend
```

Add Sensu agents to clusters

Each Sensu agent should have the following entries in `/etc/sensu/agent.yml` to ensure the agent is aware of all cluster members. This allows the agent to reconnect to a working backend if the backend it is currently connected to goes into an unhealthy state.

Here is an example backend-url configuration for all agents connecting to the cluster over WebSocket:

```
backend-url:
  - "ws://10.0.0.1:8081"
  - "ws://10.0.0.2:8081"
  - "ws://10.0.0.3:8081"
```

You should now have a highly available Sensu cluster! Confirm cluster health and try other cluster management commands with sensuctl.

Manage and monitor clusters with sensuctl

Sensuctl includes several commands to help you manage and monitor your cluster. Run `sensuctl cluster -h` for additional help information.

Get cluster health status

Get cluster health status and etcd alarm information:

```
sensuctl cluster health
```

The cluster health response will list the health status for each cluster member, similar to this example:

```
      ID          Name                      Error                  Healthy
________________ _____________________ ____________________________ _________
a32e8f613b529ad4  backend-1.example.com                                 true
c3d9f4b8d0dd1ac9  backend-2.example.com  dial tcp 10.0.0.2:2379: connect: connection refused  false
c8f63ae435a5e6bf  backend-3.example.com                                 true
```

Add a cluster member

To add a new member node to an existing cluster:

1. Configure the new node's store in its `/etc/sensu/backend.yml` configuration file. For the new node `backend-4.example.com` with IP address `10.0.0.4`:

```
etcd-advertise-client-urls: "https://10.0.0.4:2379"
etcd-listen-client-urls: "https://10.0.0.4:2379"
etcd-listen-peer-urls: "https://0.0.0.0:2380"
etcd-initial-cluster: "backend-1.example.com=https://10.0.0.1:2380,backend-
2.example.com=https://10.0.0.2:2380,backend-
3.example.com=https://10.0.0.3:2380,backend-
4.example.com=https://10.0.0.4:2380"
etcd-initial-advertise-peer-urls: "https://10.0.0.4:2380"
```

```
etcd-initial-cluster-state: "existing"
etcd-initial-cluster-token: "unique_token_for_this_cluster"
etcd-name: "backend-4.example.com"
```

> ***NOTE**: To make sure the new member is added to the correct cluster, specify the same `etcd-initial-cluster-token` value that you used for the other members in the cluster.*
>
> *Also, when you are adding a cluster member, make sure the `etcd-initial-cluster-state` value is `existing`, **not** `new`.*

2. Run the sensuctl command to add the new cluster member:

```
sensuctl cluster member-add backend-4.example.com https://10.0.0.4:2380
```

You will receive a sensuctl response to confirm that the new member was added:

```
added member 2f7ae42c315f8c2d to cluster
```

3. Start the new backend:

```
sudo systemctl start sensu-backend
```

4. Add the new cluster member's WebSocket backend-url in `/etc/sensu/agent.yml` for all agents that connect to the cluster over WebSocket:

```
backend-url:
  - "ws://10.0.0.1:8081"
  - "ws://10.0.0.2:8081"
  - "ws://10.0.0.3:8081"
  - "ws://10.0.0.4:8081"
```

List cluster members

List the ID, name, peer URLs, and client URLs of all nodes in a cluster:

```
sensuctl cluster member-list
```

You will receive a sensuctl response that lists all cluster members:

```
      ID          Name                 Peer URLs              Client URLs
_________________ _____________________ ______________________ ______________________

a32e8f613b529ad4  backend-1.example.com  https://10.0.0.1:2380  https://10.0.0.1:2379
c3d9f4b8d0dd1ac9  backend-2.example.com  https://10.0.0.2:2380  https://10.0.0.2:2379
c8f63ae435a5e6bf  backend-3.example.com  https://10.0.0.3:2380  https://10.0.0.3:2379
2f7ae42c315f8c2d  backend-4.example.com  https://10.0.0.4:2380  https://10.0.0.4:2379
```

Remove a cluster member

Remove a faulty or decommissioned member node from a cluster:

```
sensuctl cluster member-remove 2f7ae42c315f8c2d
```

You will receive a sensuctl response to confirm that the cluster member was removed:

```
Removed member 2f7ae42c315f8c2d from cluster
```

Replace a faulty cluster member

To replace a faulty cluster member to restore a cluster's health:

1. Get cluster health status and etcd alarm information:

```
```

```
sensuctl cluster health
```

In the response, for a faulty cluster member, the Error column will include an error message and the Healthy column will list `false`. In this example, the response indicates that cluster member `backend-4` is faulty:

```
     ID          Name                      Error                  Healthy
_________________ _____________________ ____________________________________ _________
a32e8f613b529ad4  backend-1.example.com                                          true
c3d9f4b8d0dd1ac9  backend-2.example.com                                          true
c8f63ae435a5e6bf  backend-3.example.com                                          true
2f7ae42c315f8c2d  backend-4.example.com  dial tcp 10.0.0.4:2379: connect: connection refused  false
```

2. Remove the faulty cluster member — in this example, `backend-4` — using its ID. Removing the faulty cluster member prevents the cluster size from growing.

```
sensuctl cluster member-remove 2f7ae42c315f8c2d
```

The response should indicate that the cluster member was removed:

```
Removed member 2f7ae42c315f8c2d from cluster
```

3. Follow the steps in Add a cluster member to configure the replacement cluster member.

> ***NOTE**: You can use the same name and IP address as the removed faulty member for the replacement cluster member. When updating the replacement member's backend configuration file, make sure the `etcd-initial-cluster-state` value is `existing`, **not** `new`.*

If replacing the faulty cluster member does not resolve the problem, read the etcd operations guide for more information.

Update a cluster member

Update the peer URLs of a member in a cluster:

```
sensuctl cluster member-update c8f63ae435a5e6bf https://10.0.0.4:2380
```

You will receive a sensuctl response to confirm that the cluster member was updated:

```
Updated member with ID c8f63ae435a5e6bf in cluster
```

Cluster security

Read Secure Sensu for information about cluster security.

Use an external etcd cluster

***WARNING**: You must update the example configuration for external etcd with an explicit, non-default configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

To use Sensu with an external etcd cluster, you must have etcd 3.3.2 or newer. To stand up an external etcd cluster, follow etcd's clustering guide using the same store configuration. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

Configure key space access

Follow these steps to configure read and write access to the `/sensu.io/` key space for your users so you can initialize a backend that uses etcd authentication.

1. Add the `sensu` user:

```
etcdctl user add sensu
```

2. Enter the `sensu` user password when prompted.

3. Create the `sensu_readwrite` role:

```
etcdctl role add sensu_readwrite
```

4. Grant read/write permissions to the `sensu_readwrite` role under the `/sensu.io/` key space:

```
etcdctl role grant-permission sensu_readwrite readwrite --from-key
'/sensu.io/'
```

5. Grant the `sensu_readwrite` role to the `sensu` user:

```
etcdctl user grant-role sensu sensu_readwrite
```

6. Confirm that the grant is configured correctly:

```
/opt/etcd/etcdctl user get USERNAME --detail
```

You should see the following output:

```
User: USERNAME

Role sensu_readwrite
KV Read:
  [/sensu.io/, <open ended>
KV Write:
  [/sensu.io/, <open ended>
```

Etcd does not enable authentication by default, so additional configuration may be needed before etcd will enforce these controls. See the etcd operators documentation for details.

Start etcd

In this example, you will enable client-to-server and peer communication authentication using self-signed TLS certificates. To start etcd for `backend-1.example.com` based on the three-node configuration example:

```
etcd \
--listen-client-urls "https://10.0.0.1:2379" \
--advertise-client-urls "https://10.0.0.1:2379" \
--listen-peer-urls "https://10.0.0.1:2380" \
--initial-cluster "backend-1.example.com=https://10.0.0.1:2380,backend-
2.example.com=https://10.0.0.2:2380,backend-3.example.com=https://10.0.0.3:2380" \
--initial-advertise-peer-urls "https://10.0.0.1:2380" \
--initial-cluster-state "new" \
--name "backend-1.example.com" \
--trusted-ca-file=./ca.pem \
--cert-file=./backend-1.example.com.pem \
--key-file=./backend-1.example.com-key.pem \
--client-cert-auth \
--peer-trusted-ca-file=./ca.pem \
--peer-cert-file=./backend-1.example.com.pem \
--peer-key-file=./backend-1.example.com-key.pem \
--peer-client-cert-auth \
--auto-compaction-mode revision \
--auto-compaction-retention 2
```

***NOTE**: Without the `auto-compaction-mode` and `auto-compaction-retention` flags, your database may quickly reach etcd's maximum database size limit.*

Tell Sensu to use this external etcd data source by adding the `sensu-backend` flag `--no-embed-etcd` to the original configuration and the path to a client certificate created using your CA:

```
sensu-backend start \
--etcd-trusted-ca-file=./ca.pem \
```

```
--etcd-cert-file=./backend-1.example.com.pem \
--etcd-key-file=./backend-1.example.com-key.pem \
--etcd-client-urls='https://10.0.0.1:2379 https://10.0.0.2:2379
https://10.0.0.3:2379' \
--no-embed-etcd
```

***NOTE**: The etcd and sensu-backend certificates must share a CA, and the `etcd-client-urls` value must be a space-delimited list or a YAML array.*

Authenticate with username and password for external etcd

Managed database services (database-as-a-service, or DBaaS) often support external etcd authentication via username and password rather than client certificates.

To use username and password authentication to connect to external etcd, add the `SENSU_BACKEND_ETCD_CLIENT_USERNAME` and `SENSU_BACKEND_ETCD_CLIENT_PASSWORD` environment variables to the environment file.Replace `<your_username>` and `<your_password>` with the username and password you use for your external etcd provider:

```
SENSU_BACKEND_ETCD_CLIENT_USERNAME=<your_username>
SENSU_BACKEND_ETCD_CLIENT_PASSWORD=<your_password>
```

Read Configuration via environment variables to learn how to create and save environment variables.

The `SENSU_BACKEND_ETCD_CLIENT_USERNAME` and `SENSU_BACKEND_ETCD_CLIENT_PASSWORD` environment variables do not have corresponding configuration flags.To use username/passsword authentication for external etcd, you must configure these environment variables in the environment file.

Migrate from embedded etcd to external etcd

To migrate from embedded etcd to external etcd, first decide whether you need to migrate all of your etcd data or just your Sensu configurations.

If you need to migrate all etcd data, you must create an etcd snapshot.Use the snapshot to restore you entire cluster after setting up the new external cluster.

If you need to migrate only your Sensu configuration, you can use sensuctl dump to create a backup and use sensuctl create to import your configuration to the new external cluster.

> ***NOTE**: The sensuctl dump command does not export user passwords, and sensuctl create does not restore API keys from a sensuctl dump backup.For this reason, you must use the etcd snapshot and restore process to migrate your entire embedded cluster to external etcd.*

After you create the backups you need, follow Use an external etcd cluster to configure Sensu to use the external cluster as your datastore.

Troubleshoot clusters

Failure modes

Read the etcd failure modes documentation for information about cluster failure modes.

Disaster recovery

For external etcd, follow the etcd recovery guide for disaster recovery.

For embedded etcd, follow Back up and recover resources with sensuctl for disaster recovery.

Redeploy a cluster

To redeploy a cluster due to an issue like loss of quorum among cluster members, etcd corruption, or hardware failure, read Remove and redeploy a cluster.

Multi-cluster visibility with federation

> ***COMMERCIAL FEATURE**: Access federation in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu's enterprise/federation/v1 API endpoints allow you to register external clusters, gain single-pane-of-glass visibility into the health of your infrastructure and services across multiple distinct Sensu instances within the web UI, and mirror your changes in one cluster to follower clusters. This is useful when you want to provide a single entry point for Sensu users who need to manage monitoring across multiple distinct physical data centers, cloud regions, or providers.

After you configure federation, you can also create, update, and delete clusters using sensuctl create, edit, and delete commands.

Federation is not enabled by default. You must create a cluster resource for the federation cluster and register it.

Only cluster administrators can register a new cluster, but every user can query the list of clusters.

Complete federation of multiple Sensu instances relies on a combination of features:

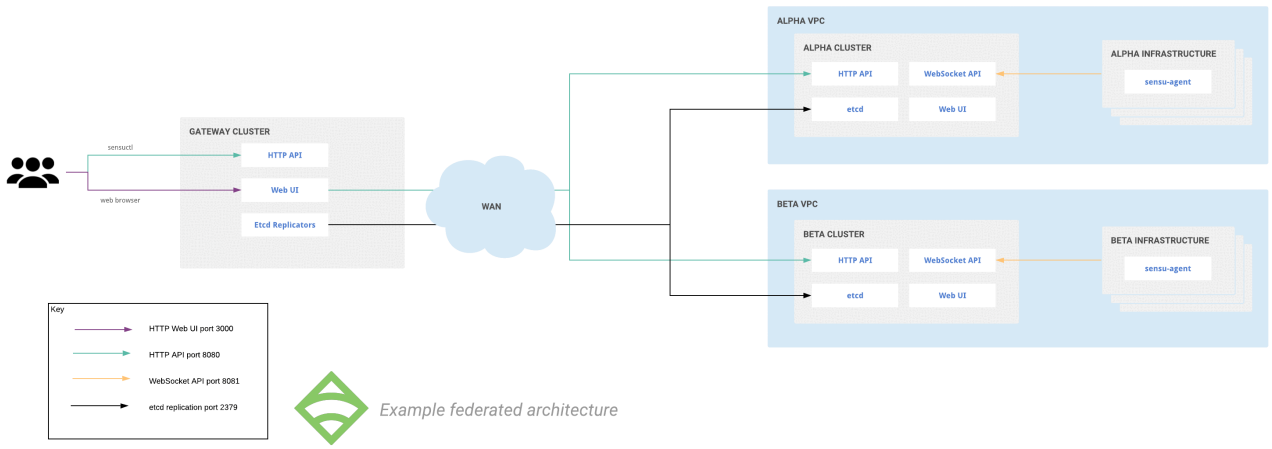
Feature	Purpose in federation
JSON Web Token (JWT) authentication	Cross-cluster token authentication using asymmetric key encryption
etcd replicators	Replicate RBAC policy across clusters and namespaces
Federation Gateway and APIs	Configure federation access for cross-cluster visibility in web UI

Follow the example in this guide to configure these features. The example assumes you wish to federate three named Sensu clusters:

Cluster name	Hostname
`gateway`	sensu.gateway.example.com
`alpha`	sensu.alpha.example.com
`beta`	sensu.beta.example.com

In this example, the `gateway` cluster will be the entry point for operators to manage Sensu resources in the `alpha` and `beta` clusters. This guide assumes a single sensu-backend in each cluster, but named clusters composed of multiple sensu-backends are supported.

This diagram depicts the federation relationship documented in this guide:



Example federated architecture

Complete the steps in this guide to browse events, entities, checks, and other resources in the

`gateway`, `alpha`, and `beta` clusters from the `gateway` cluster web UI.

Configure backends for TLS

Because federation depends on communication with multiple disparate clusters, working TLS is required for successful federated operation.

To ensure that cluster members can validate each other, certificates for each cluster member should include the IP addresses or hostnames specified in the values of sensu-backend `etcd-advertise-client-urls`, `etcd-advertise-peer-urls`, and `etcd-initial-advertise-peer-urls` parameters.In addition to the certificate's Common Name (CN), Subject Alternative Names (SANs) are also honored for validation.

> ***NOTE**: As of Go 1.15, certificates must include their CN as an SAN field.To prevent connection errors, follow Generate certificates to make sure your certificates' SAN fields include their CNs.*

To continue with this guide, make sure you have the required TLS credentials in place:

- A PEM-formatted X.509 certificate and corresponding private key copied to each cluster member.
- A corresponding certificate authority (CA) certificate chain copied to each cluster member.

If you don't have existing infrastructure for issuing certificates, read Generate certificates for our recommended self-signed certificate issuance process.

This prerequisite extends to configuring the following Sensu backend etcd parameters:

Backend property	Description
`etcd-cert-file`	Path to certificate used for TLS on etcd client/peer communications (for example, `/etc/sensu/tls/backend-1.example.com.pem`.
`etcd-key-file`	Path to key corresponding with `etcd-cert-file` certificate (for example, `/etc/sensu/tls/backend-1-key.example.com.pem`.
`etcd-trusted-ca-file`	Path to CA certificate chain file (for example, `/etc/sensu/tls/ca.pem`. This CA certificate chain must be usable to validate certificates for all backends in the federation.

`etcd-client-cert-auth`	Enforces certificate validation to authenticate etcd replicator connections. Set to `true` to secure etcd communication.
`etcd-advertise-client-urls`	List of https URLs to advertise for etcd replicators, accessible by other backends in the federation (for example, `https://sensu.beta.example.com:2379`).
`etcd-listen-client-urls`	List of https URLs to listen on for etcd replicators (for example, `https://0.0.0.0:2379` to listen on port 2379 across all ipv4 interfaces).

***WARNING**: You must provide an explicit, non-default etcd configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

This includes providing non-default values for the `etcd-advertise-client-urls` and `etcd-listen-client-urls` backend parameters and creating a certificate and key for the `etcd-cert-file` and `etcd-key-file` values. The default values are not suitable for use under federation.

Configure shared token signing keys

Whether federated or standalone, Sensu backends issue JSON Web Tokens (JWTs) to users upon successful authentication. These tokens include a payload that describes the username and group affiliations. The payload is used to determine permissions based on the configured RBAC policy.

In a federation of Sensu backends, each backend needs to have the same public/private key pair. These asymmetric keys are used to crypotgraphically vouch for the user's identity in the JWT payload. Using shared JWT keys enables clusters to grant users access to Sensu resources according to their local policies but without requiring user resources to be present uniformly across all clusters in the federation.

By default, a Sensu backend automatically generates an asymmetric key pair for signing JWTs and stores it in the etcd database. When configuring federation, you must generate keys as files on disk so they can be copied to all backends in the federation.

1. Use the `openssl` command line tool to generate a P-256 elliptic curve private key:

```
openssl ecparam -genkey -name prime256v1 -noout -out jwt_private.pem
```

2. Generate a public key from the private key:

```
openssl ec -in jwt_private.pem -pubout -out jwt_public.pem
```

3. Save the JWT keys in `/etc/sensu/certs` on each cluster backend.

4. Add the `jwt-private-key-file` and `jwt-public-key-file` attributes in `/etc/sensu/backend.yml` and specify the paths to the JWT private and public keys:

```
jwt-private-key-file: /etc/sensu/certs/jwt_private.pem
jwt-public-key-file: /etc/sensu/certs/jwt_public.pem
```

5. Restart the Sensu backend so that your settings take effect:

```
sudo systemctl restart sensu-backend
```

Add a user and a cluster role binding

To test your configuration, provision a user and a cluster role binding in the `gateway` cluster.

1. Confirm that sensuctl is configured to communicate with the `gateway` cluster:

```
sensuctl config view
```

The response will list the active configuration:

```
=== Active Configuration
API URL:   https://sensu.gateway.example.com:8080
Namespace: default
Format:    tabular
```

```
Username:  admin
```

2. Create a `federation-viewer` user:

```
sensuctl user create federation-viewer --interactive
```

3. When prompted for username and groups, press enter.

4. When prompted for password, enter a password for the `federation-viewer` user.Make a note of the password you entered — you'll use it to log in to the web UI after you configure RBAC policy replication and registered clusters into your federation.

   This creates the following user:

   TEXT

```
username: federation-viewer
disabled: false
```

   TEXT

```
{
  "username": "federation-viewer",
  "disabled": false
}
```

5. Grant the `federation-viewer` user read-only access with a cluster role binding for the built-in `view` cluster role:

```
sensuctl cluster-role-binding create federation-viewer-readonly --cluster-
role=view --user=federation-viewer
```

   This command creates the following cluster role binding resource definition:

   YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: federation-viewer-readonly
spec:
  role_ref:
    name: view
    type: ClusterRole
  subjects:
  - name: federation-viewer
    type: User
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "federation-viewer-readonly"
  },
  "spec": {
    "role_ref": {
      "name": "view",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "federation-viewer",
        "type": "User"
      }
    ]
  }
}
```

Create etcd replicators

Etcd replicators use the etcd make-mirror utility for one-way replication of Sensu RBAC policy resources. This allows you to centrally define RBAC policy on the `gateway` cluster and replicate RBAC resources to other clusters in the federation (`alpha` and `beta`), ensuring consistent permissions for Sensu users across multiple clusters via the `gateway` web UI.

1. Configure one etcd replicator per cluster for each RBAC policy resource, across all namespaces, for each backend in the federation.

   > ***NOTE**: Create a replicator for each resource type you want to replicate. Replicating `namespace` resources will **not** replicate the Sensu resources that belong to those namespaces.*
   >
   > *The etcd replicators reference includes examples you can follow for `Role` , `RoleBinding` , `ClusterRole` , and `ClusterRoleBinding` resources.*

   In this example, the following etcd replicator resources will replicate ClusterRoleBinding resources from the `gateway` cluster to the two target clusters:

   YML

   ```
   ---
   api_version: federation/v1
   type: EtcdReplicator
   metadata:
     name: AlphaClusterRoleBindings
   spec:
     ca_cert: "/etc/sensu/certs/ca.pem"
     cert: "/etc/sensu/certs/gateway.pem"
     key: "/etc/sensu/certs/gateway-key.pem"
     url: https://sensu.alpha.example.com:2379
     api_version: core/v2
     resource: ClusterRoleBinding
     replication_interval_seconds: 30
   ```

   JSON

   ```
   {
     "api_version": "federation/v1",
     "type": "EtcdReplicator",
     "metadata": {
       "name": "AlphaClusterRoleBindings"
   ```

```
  },
  "spec": {
    "ca_cert": "/etc/sensu/certs/ca.pem",
    "cert": "/etc/sensu/certs/gateway.pem",
    "key": "/etc/sensu/certs/gateway-key.pem",
    "url": "https://sensu.alpha.example.com:2379",
    "api_version": "core/v2",
    "resource": "ClusterRoleBinding",
    "replication_interval_seconds": 30
  }
}
```

YML

```
---
api_version: federation/v1
type: EtcdReplicator
metadata:
  name: BetaClusterRoleBindings
spec:
  ca_cert: "/etc/sensu/certs/ca.pem"
  cert: "/etc/sensu/certs/gateway.pem"
  key: "/etc/sensu/certs/gateway-key.pem"
  url: https://sensu.beta.example.com:2379
  api_version: core/v2
  resource: ClusterRoleBinding
  replication_interval_seconds: 30
```

JSON

```
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "BetaClusterRoleBindings"
  },
  "spec": {
    "ca_cert": "/etc/sensu/certs/ca.pem",
    "cert": "/etc/sensu/certs/gateway.pem",
    "key": "/etc/sensu/certs/gateway-key.pem",
    "url": "https://sensu.beta.example.com:2379",
```

```
    "api_version": "core/v2",
    "resource": "ClusterRoleBinding",
    "replication_interval_seconds": 30
  }
}
```

2. Run `sensuctl config view` and verify that `sensuctl` is configured to talk to a `gateway` cluster API.Reconfigure `sensuctl` if needed.

3. Save the `AlphaClusterRoleBindings` and `BetaClusterRoleBindings` EtcdReplicator definitions to a file (for example, `etcdreplicators.yml` or `etcdreplicators.json`).

4. Use `sensuctl create -f` to apply the `AlphaClusterRoleBindings` and `BetaClusterRoleBindings` EtcdReplicator definitions to the `gateway` cluster:

   SHELL

   ```
   sensuctl create -f etcdreplicators.yml
   ```

   SHELL

   ```
   sensuctl create -f etcdreplicators.json
   ```

5. Verify that the EtcdReplicator resource is working as expected: reconfigure the sensuctl backend URL to communicate with the `alpha` and `beta` clusters and run the following command for each:

   ```
   sensuctl cluster-role-binding info federation-viewer-readonly
   ```

   The `federation-viewer-readonly` binding you created in the previous section should be listed in the output from each cluster:

   ```
   === federation-viewer-readonly
   Name:         federation-viewer-readonly
   Cluster Role: view
   Subjects:
   ```

```
    Users:      federation-viewer
```

Register clusters

Clusters must be registered to become visible in the web UI.Each registered cluster must have a name and a list of one or more cluster member URLs corresponding to the backend REST API.

> ***NOTE**: Individual cluster resources may list the API URLs for a single stand-alone backend or multiple backends that are members of the same etcd cluster.Creating a cluster resource that lists multiple backends that do not belong to the same cluster will result in unexpected behavior.*

Register a single cluster

With `sensuctl` configured for the `gateway` cluster, run `sensuctl create` on the yaml or JSON below to register cluster `alpha`:

YML

```
api_version: federation/v1
type: Cluster
metadata:
  name: alpha
spec:
  api_urls:
  - https://sensu.alpha.example.com:8080
```

JSON

```
{
  "api_version": "federation/v1",
  "type": "Cluster",
  "metadata": {
    "name": "alpha"
  },
  "spec": {
    "api_urls": [
      "https://sensu.alpha.example.com:8080"
```

```
    ]
  }
}
```

Register additional clusters

With `sensuctl` configured for `gateway` cluster, run `sensuctl create` on the yaml or JSON below to register an additional cluster and define the name as `beta`:

YML

```
api_version: federation/v1
type: Cluster
metadata:
  name: beta
spec:
  api_urls:
  - https://sensu.beta.example.com:8080
```

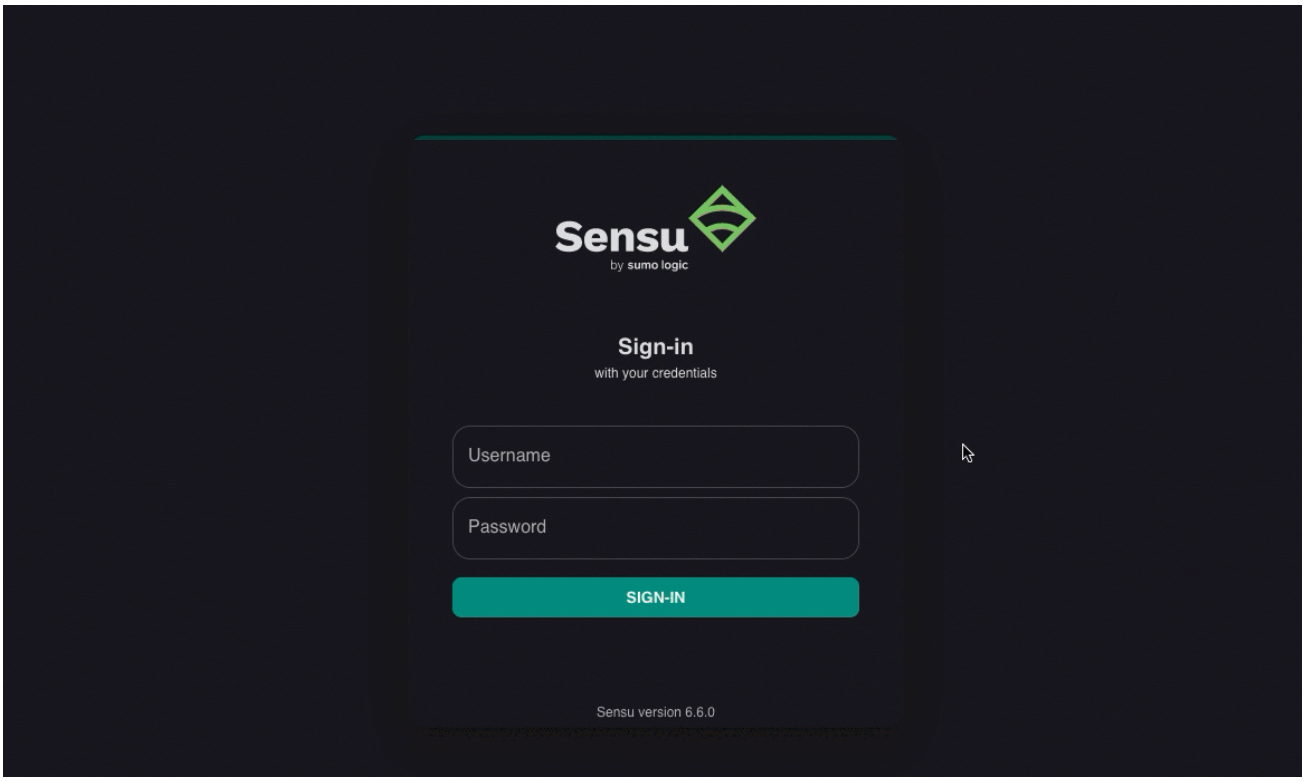
JSON

```
{
  "api_version": "federation/v1",
  "type": "Cluster",
  "metadata": {
    "name": "beta"
  },
  "spec": {
    "api_urls": [
      "https://sensu.alpha.example.com:8080"
    ]
  }
}
```

***NOTE**: When logging into the `gateway` cluster web UI, any namespaces, entities, events, and other resources specific to that cluster will be labeled as `local-cluster`.*

Get a unified view of all your clusters in the web UI

After you create clusters using enterprise/federation/v1 API endpoints, you can log in to the `gateway` Sensu web UI to view them as the `federation-viewer` user. Use the namespace switcher to change between namespaces across federated clusters:



Because the `federation-viewer` user is granted only permissions provided by the built-in `view` role, this user should be able to view all resources across all clusters but should not be able to make any changes. If you haven't changed the permissions of the default `admin` user, that user should be able to view, create, delete, and update resources across all clusters.

Next steps

Learn more about configuring RBAC policies in our RBAC reference documentation.

Scale Sensu Go with Enterprise datastore

***COMMERCIAL FEATURE**: Access the datastore feature in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu Go's datastore feature enables scaling your monitoring to many thousands of events per second.

For each unique entity/check pair, Sensu records the latest event object in its datastore. By default, Sensu uses the embedded etcd datastore for event storage. The embedded etcd datastore helps you get started, but as the number of entities and checks in your Sensu implementation grows, so does the rate of events being written to the datastore. In a clustered deployment of etcd, whether embedded or external to Sensu, each event received by a member of the cluster must be replicated to other members, increasing network and disk IO utilization.

Our team documented configuration and testing of Sensu running on bare metal infrastructure in the sensu/sensu-perf project. This configuration comfortably handled 12,000 Sensu agent connections (and their keepalives) and processed more than 8,500 events per second.

This rate of events should be sufficient for many installations but assumes an ideal scenario where Sensu backend nodes use direct-attached, dedicated non-volatile memory express (NVMe) storage and are connected to a dedicated LAN. Deployments on public cloud providers are not likely to achieve similar results due to sharing both disk and network bandwidth with other tenants. Adhering to the cloud provider's recommended practices may also become a factor because many operators are inclined to deploy a cluster across multiple availability zones. In such a deployment cluster, communication happens over shared WAN links, which are subject to uncontrolled variability in throughput and latency.

The Enterprise datastore can help operators achieve much higher rates of event processing and minimize the replication communication between etcd peers. The `sensu-perf` test environment comfortably handles 40,000 Sensu agent connections (and their keepalives) and processes more than 36,000 events per second under ideal conditions.

***IMPORTANT**: PostgreSQL configuration file locations differ depending on platform. The steps in this guide use common paths for RHEL-family distributions, but the files may be stored elsewhere on your system. Learn more about PostgreSQL configuration file locations.*

Prerequisites

- Database server running Postgres 9.5 or later
- Postgres database (or administrative access to create one)
- Postgres user with permissions to the database (or administrative access to create such a user)
- Licensed Sensu Go backend

For optimal performance, we recommend the following PostgreSQL configuration parameters and settings as a starting point for your `postgresql.conf` file:

```
max_connections = 200

shared_buffers = 10GB

maintenance_work_mem = 1GB

vacuum_cost_delay = 10ms
vacuum_cost_limit = 10000

bgwriter_delay = 50ms
bgwriter_lru_maxpages = 1000

max_worker_processes = 8
max_parallel_maintenance_workers = 2
max_parallel_workers_per_gather = 2
max_parallel_workers = 8

synchronous_commit = off

wal_sync_method = fdatasync
wal_writer_delay = 5000ms
max_wal_size = 5GB
min_wal_size = 1GB

checkpoint_completion_target = 0.9

autovacuum_naptime = 10s
```

```
autovacuum_vacuum_scale_factor = 0.05
autovacuum_analyze_scale_factor = 0.025
```

Adjust the parameters and settings as needed based on your hardware and the performance you observe.Read the PostgreSQL parameters documentation for information about setting parameters.

Configure Postgres

Before Sensu can start writing events to Postgres, you need a database and an account with permissions to write to that database. To provide consistent event throughput, we recommend exclusively dedicating your Postgres instance to storage of Sensu events.

If you have administrative access to Postgres, you can create the database and user.

1. Change to the postgres user and open the Postgres prompt (`postgres=#`):

   ```
   sudo -u postgres psql
   ```

2. Create the `sensu_events` database:

   ```
   CREATE DATABASE sensu_events;
   ```

   PostgreSQL will return a confirmation message: `CREATE DATABASE` .

3. Create the `sensu` role with a password:

   ```
   CREATE USER sensu WITH ENCRYPTED PASSWORD 'mypass';
   ```

   PostgreSQL will return a confirmation message: `CREATE ROLE` .

4. Grant the `sensu` role all privileges for the `sensu_events` database:

   ```
   GRANT ALL PRIVILEGES ON DATABASE sensu_events TO sensu;
   ```

PostgreSQL will return a confirmation message: `GRANT`.

5. Type `\q` to exit the PostgreSQL prompt.

With this configuration complete, PostgreSQL will have a `sensu_events` database for storing Sensu events and a `sensu` user with permissions to that database.

By default, the Postgres user you've just added will not be able to authenticate via password, so you'll also need to make a change to the `pg_hba.conf` file. The required change will depend on how Sensu will connect to Postgres. In this case, you'll configure Postgres to allow the `sensu` user to connect to the `sensu_events` database from any host using an md5-encrypted password:

1. Make a copy of the current `pg_hba.conf` file:

```
sudo cp /var/lib/pgsql/data/pg_hba.conf /var/tmp/pg_hba.conf.bak
```

2. Give the Sensu user permissions to connect to the `sensu_events` database from any IP address:

```
echo 'host sensu_events sensu 0.0.0.0/0 md5' | sudo tee -a
/var/lib/pgsql/data/pg_hba.conf
```

3. Restart the postgresql service to activate the `pg_hba.conf` changes:

```
sudo systemctl restart postgresql
```

With this configuration complete, you can configure Sensu to store events in your Postgres database.

To secure communication between Sensu and the PostgreSQL event store using certificate authentication, read Secure PostgreSQL.

Configure Sensu

If your Sensu backend is already licensed, the configuration for routing events to Postgres is relatively straightforward.Create a `PostgresConfig` resource that describes the database connection as a data source name (DSN):

YML

```
---
type: PostgresConfig
api_version: store/v1
metadata:
  name: postgres01
spec:
  dsn: "postgresql://sensu:mypass@10.0.2.15:5432/sensu_events?sslmode=disable"
  pool_size: 20
```

JSON

```
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "dsn": "postgresql://sensu:mypass@10.0.2.15:5432/sensu_events",
    "pool_size": 20
  }
}
```

Save this configuration as `my-postgres.yml` or `my-postgres.json` and install it with `sensuctl`:

SHELL

```
sensuctl create --file my-postgres.yml
```

SHELL

```
sensuctl create --file my-postgres.json
```

The Sensu backend is now configured to use Postgres for event storage!

In the web UI and in sensuctl, event history will appear incomplete. When Postgres configuration is provided and the backend successfully connects to the database, etcd event history is not migrated. New events will be written to Postgres as they are processed, with the Postgres datastore ultimately being brought up to date with the current state of your monitored infrastructure.

Aside from event history, which is not migrated from etcd, there's no observable difference when using Postgres as the event store, and neither interface supports displaying the PostgresConfig type.

To verify that the change was effective and your connection to Postgres was successful, look at the sensu-backend log:

```
{"component":"store","level":"warning","msg":"trying to enable external event store","time":"2019-10-02T23:31:38Z"}
{"component":"store","level":"warning","msg":"switched event store to postgres","time":"2019-10-02T23:31:38Z"}
```

You can also use `psql` to verify that events are being written to the `sensu_events` database.

1. Change to the postgres user and open the Postgres prompt (`postgres=#`):

   ```
   sudo -u postgres psql
   ```

2. Connect to the `sensu_events` database:

   ```
   \c sensu_events
   ```

   PostgreSQL will return a confirmation message:

   ```
   You are now connected to database "sensu_events" as user "postgres".
   ```

   The prompt will change to `sensu_events=#` .

3. List the tables in the `sensu_events` database:

```
\dt
```

PostgreSQL will list the tables:

```
             List of relations
 Schema |       Name        | Type  | Owner
--------+-------------------+-------+-------
 public | events            | table | sensu
 public | migration_version | table | sensu
(2 rows)
```

4. Request a list of all entities reporting keepalives:

```
select sensu_entity from events where sensu_check = 'keepalive';
```

PostgreSQL will return a list of the entities:

```
 sensu_entity
--------------
 i-414141
 i-424242
 i-434343
(3 rows)
```

Revert to the built-in datastore

If you want to revert to the default etcd event store, delete the PostgresConfig resource. In this example, `my-postgres.yml` or `my-postgres.json` contain the same configuration you used to configure the Enterprise event store earlier in this guide:

SHELL

```
sensuctl delete --file my-postgres.yml
```

SHELL

```
sensuctl delete --file my-postgres.json
```

To verify that the change was effective, look for messages similar to these in the sensu-backend log:

```
{"component":"store","level":"warning","msg":"store configuration
deleted","store":"/sensu.io/api/enterprise/store/v1/provider/postgres01","time":"201
9-10-02T23:29:06Z"}
{"component":"store","level":"warning","msg":"switched event store to
etcd","time":"2019-10-02T23:29:06Z"}
```

Similar to enabling PostgreSQL, switching back to the etcd datastore does not migrate current observability event data from one store to another. The web UI or sensuctl output may list outdated events until the etcd datastore catches up with the current state of your monitored infrastructure.

Configure Postgres streaming replication

Postgres supports an active standby with streaming replication. Configure streaming replication to replicate all Sensu events written to the primary Postgres server to the standby server.

Follow the steps in this section to create and add the replication role, set streaming replication configuration parameters, bootstrap the standby host, and confirm successful Postgres streaming replication.

Create and add the replication role

If you have administrative access to Postgres, you can create the replication role. Follow these steps to create and add the replication role on the **primary** Postgres host:

1. Change to the postgres user and open the Postgres prompt (`postgres=#`):

```
sudo -u postgres psql
```

2. Create the `repl` role:

```
CREATE ROLE repl PASSWORD '<your-password>' LOGIN REPLICATION;
```

PostgreSQL will return a confirmation message: `CREATE ROLE`.

3. Type `\q` to exit the PostgreSQL prompt.

4. Add the replication role to `pg_hba.conf` using an md5-encrypted password. Make a copy of the current `pg_hba.conf`:

```
sudo cp /var/lib/pgsql/data/pg_hba.conf /var/tmp/pg_hba.conf.bak
```

5. In the following command, replace `<standby_ip>` with the IP address of your standby Postgres host and run the command:

```
export STANDBY_IP=<standby-ip>
```

6. Give the repl user permissions to replicate from the standby host:

```
echo "host replication repl ${STANDBY_IP}/32 md5" | sudo tee -a
/var/lib/pgsql/data/pg_hba.conf
```

7. Restart the PostgreSQL service to activate the `pg_hba.conf` changes:

```
sudo systemctl restart postgresql
```

Set streaming replication configuration parameters

Follow these steps to set streaming replication configuration parameters on the **primary** Postgres host:

1. Make a copy of the `postgresql.conf`:

```
sudo cp -a /var/lib/pgsql/data/postgresql.conf
/var/lib/pgsql/data/postgresql.conf.bak
```

2. Append the necessary configuration options.

```
echo 'wal_level = replica' | sudo tee -a /var/lib/pgsql/data/postgresql.conf
```

3. Set the maximum number of concurrent connections from the standby servers:

```
echo 'max_wal_senders = 5' | sudo tee -a /var/lib/pgsql/data/postgresql.conf
```

4. To prevent the primary server from removing the WAL segments required for the standby server before shipping them, set the minimum number of segments retained in the `pg_xlog` directory:

```
echo 'wal_keep_segments = 32' | sudo tee -a
/var/lib/pgsql/data/postgresql.conf
```

   At minimum, the number of `wal_keep_segments` should be larger than the number of segments generated between the beginning of online backup and the startup of streaming replication.

   > ***NOTE**: If you enable WAL archiving to an archive directory accessible from the standby, this step may not be necessary.*

5. Restart the PostgreSQL service to activate the `postgresql.conf` changes:

```
sudo systemctl restart postgresql
```

Bootstrap the standby host

Follow these steps to bootstrap the standby host on the **standby** Postgres host:

1. If the standby host has ever run Postgres, stop Postgres and empty the data directory:

```
sudo systemctl stop postgresql
```

```
sudo mv /var/lib/pgsql/data /var/lib/pgsql/data.bak
```

2. Make the standby data directory:

```
sudo install -d -o postgres -g postgres -m 0700 /var/lib/pgsql/data
```

3. In the following command, replace `<primary_ip>` with the IP address of your primary Postgres host and run the command:

```
export PRIMARY_IP=<primary_ip>
```

4. Bootstrap the standby data directory:

```
sudo -u postgres pg_basebackup -h $PRIMARY_IP -D /var/lib/pgsql/data -P -U
repl -R --wal-method=stream
```

5. Enter your password at the Postgres prompt:

```
Password:
```

After you enter your password, PostgreSQL will list database copy progress:

```
30318/30318 kB (100%), 1/1 tablespace
```

Confirm replication

Follow these steps to confirm replication:

1. On the **primary** Postgres host, remove primary-only configurations:

   ```
   sudo sed -r -i.bak '/^(wal_level|max_wal_senders|wal_keep_segments).*/d'
   /var/lib/pgsql/data/postgresql.conf
   ```

2. Start the PostgreSQL service:

   ```
   sudo systemctl start postgresql
   ```

3. Check the primary host commit log location:

   ```
   sudo -u postgres psql -c "select pg_current_wal_lsn()"
   ```

   PostgreSQL will list the primary host commit log location:

   ```
    pg_current_wal_lsn
   -------------------------
    0/3000568
   (1 row)
   ```

4. On the **standby** Postgres host, check the commit log location:

   ```
   ```

```
sudo -u postgres psql -c "select pg_last_wal_receive_lsn()"
```

```
sudo -u postgres psql -c "select pg_last_wal_replay_lsn()"
```

PostgreSQL will list the standby host commit log location:

```
 pg_last_wal_receive_lsn
-------------------------------
 0/3000568
(1 row)
```

```
 pg_last_wal_replay_lsn
-------------------------------
 0/3000568
(1 row)
```

With this configuration complete, your Sensu events will be replicated to the standby host.

Datastore reference

Sensu stores the most recent event for each entity and check pair using either an etcd (default) or PostgreSQL database.

You can access observability event data with the Sensu web UI Events page, `sensuctl event` commands, and core/v2/events API endpoints. For longer retention of observability event data, integrate Sensu with a time-series database like InfluxDB or a searchable index like ElasticSearch or Splunk.

etcd and PostgreSQL version compatibility

Sensu requires at least etcd 3.3.2 and is tested against etcd 3.5. etcd version 3.4.0 is compatible with Sensu but may result in slower performance.

Sensu supports PostgreSQL 9.5 and later, including Amazon Relational Database Service (Amazon RDS) when configured with the PostgreSQL engine.

Use default event storage

By default, Sensu uses its embedded etcd database to store configuration and event data. This embedded database allows you to get started with Sensu without deploying a complete, scalable architecture. Sensu's default embedded etcd configuration listens for unencrypted communication on ports 2379 (client requests) and 2380 (peer communication).

Sensu can be configured to disable the embedded etcd database and use one or more external etcd nodes for configuration and event storage instead. To stand up an external etcd cluster, follow etcd's clustering guide using the same store configuration. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

As your deployment grows beyond the proof-of-concept stage, review Deployment architecture for Sensu for more information about deployment considerations and recommendations for a production-ready Sensu deployment.

Scale event storage

***COMMERCIAL FEATURE**: Access enterprise-scale event storage in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu supports using an external PostgreSQL instance for event storage in place of etcd. PostgreSQL can handle significantly higher volumes of Sensu events, which allows you to scale Sensu beyond etcd's 8GB limit.

When configured with a PostgreSQL event store, Sensu connects to PostgreSQL to store and retrieve event data in place of etcd. Etcd continues to store Sensu entity and configuration data. You can access event data stored in PostgreSQL using the same Sensu web UI, API, and sensuctl processes as etcd-stored events.

Read the PostgreSQL documentation to install and configure PostgreSQL.

PostgreSQL requirements

For optimal performance, we recommend the following PostgreSQL configuration parameters and settings as a starting point for your `postgresql.conf` file:

```
max_connections = 200

shared_buffers = 10GB

maintenance_work_mem = 1GB

vacuum_cost_delay = 10ms
vacuum_cost_limit = 10000

bgwriter_delay = 50ms
bgwriter_lru_maxpages = 1000

max_worker_processes = 8
max_parallel_maintenance_workers = 2
max_parallel_workers_per_gather = 2
max_parallel_workers = 8

synchronous_commit = off
```

```
wal_sync_method = fdatasync
wal_writer_delay = 5000ms
max_wal_size = 5GB
min_wal_size = 1GB

checkpoint_completion_target = 0.9

autovacuum_naptime = 10s
autovacuum_vacuum_scale_factor = 0.05
autovacuum_analyze_scale_factor = 0.025
```

Adjust the parameters and settings as needed based on your hardware and the performance you observe.Read the PostgreSQL parameters documentation for information about setting parameters.

Configure the PostgreSQL event store

At the time when you enable the PostgreSQL event store, event data cuts over from etcd to PostgreSQL.This results in a loss of recent event history.No restarts or Sensu backend configuration changes are required to enable the PostgreSQL event store.

When you successfully enable PostgreSQL as the Sensu Go event store, the Sensu backend log will include a message like this:

```
Mar 10 17:44:45 sensu-centos sensu-backend[1365]: {"component":"store-providers","level":"warning","msg":"switched event store to postgres","time":"2020-03-10T17:44:45Z"}
```

After you install and configure PostgreSQL, configure Sensu by creating a `PostgresConfig` resource like the following example.Review the datastore specification for more information.

YML

```
---
type: PostgresConfig
api_version: store/v1
metadata:
  name: my-postgres
spec:
  batch_buffer: 0
```

```
  batch_size: 1
  batch_workers: 0
  dsn: "postgresql://user:secret@host:port/dbname"
  max_conn_lifetime: 5m
  max_idle_conns: 2
  pool_size: 20
  strict: true
  enable_round_robin: true
```

JSON

```
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true,
    "enable_round_robin": true
  }
}
```

Save your `PostgresConfig` resource definition to a file (in this example, `postgres.yml` or `postgres.json`). Then, use sensuctl configured as the admin user to activate the PostgreSQL event store.

SHELL

```
sensuctl create --file postgres.yml
```

SHELL

```
sensuctl create --file postgres.json
```

To update your Sensu PostgreSQL configuration, repeat the `sensuctl create` process. You can expect PostgreSQL status updates in the Sensu backend logs at the `warn` log level and PostgreSQL error messages in the Sensu backend logs at the `error` log level.

Use environment variables to configure PostgreSQL

The Sensu backend uses the libpq library to make connections to PostgreSQL. libpq supports a number of environment variables that can be injected into the PostgreSQL data source name (DSN). Sensu loads these environment variables at runtime using the system's environment variable file.

You can use the libpq environment variables to connect to PostgreSQL without exposing sensitive information, like usernames and passwords, in your PostgreSQL configuration. To do this, define the libpq environment variables as described in the backend reference. Sensu automatically looks up these environment variables, so you do not need to reference them in your PostgresConfig definition.

For example, to use libpq environment variables to define the PostgreSQL username, password, port, and database name in the Sensu backend environment file:

```
PGUSER=<PostgreSQL_username>
PGPASSWORD=<PostgreSQL_password>
PGPORT=5432
PGDATABASE=sensu_events
```

With these environment variables defined, the PostgreSQL configuration does not need to include the username, password, port, or database name:

YML

```yaml
---
type: PostgresConfig
api_version: store/v1
metadata:
  name: postgres_datastore
spec:
  dsn: "postgresql://postgres.example.com"
  pool_size: 20
```

```
  strict: true
```

JSON

```
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "postgres_datastore"
  },
  "spec": {
    "dsn": "postgresql://postgres.example.com",
    "pool_size": 20,
    "strict": true
  }
}
```

Disable the PostgreSQL event store

To disable the PostgreSQL event store, use `sensuctl delete` with your `PostgresConfig` resource definition file:

SHELL

```
sensuctl delete --file postgres.yml
```

SHELL

```
sensuctl delete --file postgres.json
```

The Sensu backend log will include a message to record that you successfully disabled PostgreSQL as the Sensu Go event store:

```
Mar 10 17:35:04 sensu-centos sensu-backend[1365]: {"component":"store-
providers","level":"warning","msg":"switched event store to etcd","time":"2020-03-
10T17:35:04Z"}
```

When you disable the PostgreSQL event store, event data cuts over from PostgreSQL to etcd, which results in a loss of recent event history. No restarts or Sensu backend configuration changes are required to disable the PostgreSQL event store.

Datastore specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For PostgreSQL datastore configs, the `api_version` should be `store/v1`.
required	true
type	String
example	**YML** `api_version: store/v1` **JSON** `{ "api_version": "store/v1" }`

metadata	
description	Top-level scope that contains the PostgreSQL datastore `name` and `created_by` field.
required	true
type	Map of key-value pairs

example	**YML** ```yaml metadata: name: my-postgres created_by: admin ``` **JSON** ```json { "metadata": { "name": "my-postgres", "created_by": "admin" } } ```

spec

description	Top-level map that includes the PostgreSQL datastore config spec attributes.
required	true
type	Map of key-value pairs
example	**YML** ```yaml spec: batch_buffer: 0 batch_size: 1 batch_workers: 0 dsn: 'postgresql://user:secret@host:port/dbname' max_conn_lifetime: 5m max_idle_conns: 2 pool_size: 20 strict: true enable_round_robin: true ``` **JSON**

```
{
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true,
    "enable_round_robin": true
  }
}
```

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. PostgreSQL datastore configs should always be type `PostgresConfig`.
required	true
type	String
example	**YML** `type: PostgresConfig` **JSON** `{ "type": "PostgresConfig" }`

Metadata attributes

created_by

description	Username of the Sensu user who created the datastore or last updated the datastore. Sensu automatically populates the `created_by` field when the datastore is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

name

description	PostgreSQL datastore name used internally by Sensu.
required	true
type	String
example	**YML** `name: my-postgres` **JSON** `{ "name": "my-postgres" }`

Spec attributes

batch_buffer

description	Maximum number of requests to buffer in memory. ***WARNING**: The batcher is sensitive to configuration values, and some `batch_buffer`, `batch_size`, and `batch_workers` combinations will not work optimally. We do not recommend configuring this attribute while we are testing and improving it.*
required	false
default	0
type	Integer
example	**YML** `batch_buffer: 0` **JSON** `{ "batch_buffer": 0 }`

batch_size

description	Number of requests in each PostgreSQL write transaction, as specified in the PostgreSQL configuration. ***WARNING**: The batcher is sensitive to configuration values, and some `batch_buffer`, `batch_size`, and `batch_workers` combinations will not work optimally. We do not recommend*

configuring this attribute while we are testing and improving it.

required	false
default	1
type	Integer
example	**YML** `batch_size: 1` **JSON** `{ "batch_size": 1 }`

batch_workers

description	Number of Goroutines sending data to PostgreSQL, as specified in the PostgreSQL configuration. ***WARNING**: The batcher is sensitive to configuration values, and some `batch_buffer`, `batch_size`, and `batch_workers` combinations will not work optimally. We do not recommend configuring this attribute while we are testing and improving it.*
required	false
default	Current PostgreSQL pool size
type	Integer
example	**YML** `batch_workers: 0`

JSON

```
{
  "batch_workers": 0
}
```

dsn

description	Data source name. Specified as a URL or PostgreSQL connection string. The Sensu backend uses the Go pq library, which supports a subset of the PostgreSQL libpq connection string parameters. To avoid exposing sensitive information in the `dsn` attribute, configure PostgreSQL with environment variables.
required	true
type	String

YML

```
dsn: 'postgresql://user:secret@host:port/dbname'
```

JSON

```
{
  "dsn": "postgresql://user:secret@host:port/dbname"
}
```

enable_round_robin

description	If `true`, enables round robin scheduling on PostgreSQL. Any existing round robin scheduling will stop and migrate to PostgreSQL as entities check in with keepalives. Sensu will gradually delete the existing etcd scheduler state as keepalives on the etcd scheduler keys expire over

time. Otherwise, `false`.

We recommend using PostgreSQL rather than etcd for round robin scheduling because etcd leases are not reliable enough to produce precise round robin behavior.

required	false
default	false
type	Boolean
example	**YML** `enable_round_robin: true` **JSON** `{ "enable_round_robin": true }`

max_conn_lifetime

description	Maximum time a connection can persist before being destroyed. Specify values with a numeral and a letter indicator: `s` to indicate seconds, `m` to indicate minutes, and `h` to indicate hours. For example, `1m`, `2h`, and `2h1m3s` are valid.
required	false
type	String
example	**YML** `max_conn_lifetime: 5m` **JSON** `{ "max_conn_lifetime": "5m"`

```
}
```

max_idle_conns

description	Maximum number of number of idle connections to retain.
required	false
default	`2`
type	Integer
example	**YML** `max_idle_conns: 2` **JSON** `{ "max_idle_conns": 2 }`

pool_size

description	Maximum number of connections to hold in the PostgreSQL connection pool. We recommend `20` for most instances.
required	false
default	`0` (unlimited)
type	Integer
example	**YML** `pool_size: 20`

JSON

```
{
  "pool_size": 20
}
```

strict

description	If `true`, when the PostgresConfig resource is created, configuration validation will include connecting to the PostgreSQL database and executing a query to confirm whether the connected user has permission to create database tables. Otherwise, `false`. If `strict: true`, sensu-backend will try to connect to PostgreSQL indefinitely at 5-second intervals instead of reverting to etcd after 3 attempts. We recommend setting `strict: true` in most cases. If the connection fails or the user does not have permission to create database tables, resource configuration will fail and the configuration will not be persisted. This extended configuration is useful for debugging when you are not sure whether the configuration is correct or the database is working properly.
required	false
default	false
type	Boolean
example	**YML** `strict: true` **JSON** `{ "strict": true }`

Etcd replicators reference

***COMMERCIAL FEATURE**: Access the EtcdReplicator datatype in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: EtcdReplicator is a datatype in the enterprise/federation/v1 API, which is only accessible for users who have a cluster role that permits access to replication resources.*

Etcd replicators allow you to manage role-based access control (RBAC) resources in one place and mirror the changes to follower clusters. The API sets up etcd mirrors for one-way key replication.

The EtcdReplicator datatype will not use a namespace because it applies cluster-wide. Therefore, only cluster role RBAC bindings will apply to it.

Etcd replicator examples

Use the following four examples for `Role`, `RoleBinding`, `ClusterRole`, and `ClusterRoleBinding` resources to create a full replication of RBAC policy.

***NOTE**: If you do not specify a namespace when you create a replicator, all namespaces for a given resource are replicated.*

`Role` resource example

YML

```
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: role_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
```

```
  key: /path/to/ssl/key.pem
  insecure: false
  url: http://127.0.0.1:2379
  api_version: core/v2
  resource: Role
  replication_interval_seconds: 30
```

JSON

```
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "role_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

`RoleBinding` resource example

YML

```
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: rolebinding_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  key: /path/to/ssl/key.pem
  insecure: false
```

```
  url: http://127.0.0.1:2379
  api_version: core/v2
  resource: RoleBinding
  replication_interval_seconds: 30
```

JSON

```
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "rolebinding_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "RoleBinding",
    "replication_interval_seconds": 30
  }
}
```

`ClusterRole` resource example

YML

```
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: clusterrole_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  key: /path/to/ssl/key.pem
  insecure: false
  url: http://127.0.0.1:2379
  api_version: core/v2
```

```
  resource: ClusterRole
  replication_interval_seconds: 30
```

JSON

```
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "clusterrole_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "ClusterRole",
    "replication_interval_seconds": 30
  }
}
```

`ClusterRoleBinding` resource example

YML

```
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: clusterrolebinding_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  key: /path/to/ssl/key.pem
  insecure: false
  url: http://127.0.0.1:2379
  api_version: core/v2
  resource: Role
  replication_interval_seconds: 30
```

JSON

```
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "clusterrolebinding_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "ClusterRoleBinding",
    "replication_interval_seconds": 30
  }
}
```

Critical success factors for etcd replication

Before you implement etcd replicators, review these details — they are critical to your success.

Bind your etcd listener to an external port that is *not* the default.

- Replication will not work if you bind your etcd listener to the default port.

Use only addresses that clients can route to for `etcd-client-advertise-urls`.

- If you use addresses that clients cannot route to for `etcd-client-advertise-urls`, replication may be inconsistent: it may work at first but then stop working later.

Put the certificate and key of the follower cluster in files that the leader can access.

- If the leader cannot access the follower cluster files that contain the certificate and key, replication will not work.

For self-signed certificates, supply the CA certificate in the replicator definition.

- If you have a self-signed certificate and you do not supply the CA certificate in the replicator definition, replication will not work.

If you're using insecure mode, use TLS mutual authentication.

- Never use insecure mode without TLS mutual authentication outside of a testbed.

Create a replicator for each resource type you want to replicate.

- Replicating `namespace` resources will **not** replicate the resources that belong to those namespaces.

***WARNING**: Make sure to confirm your configuration. The server will accept incorrect EtcdReplicator definitions without sending a warning. If your configuration is incorrect, replication will not work.*

Create a replicator

You can use enterprise/federation/v1 API endpoints directly or `sensuctl create` to create replicators.

When you create or update a replicator, an entry is added to the store and a new replicator process will spin up. The replicator process watches the keyspace of the resource to be replicated and replicates all keys to the specified cluster in a last-write-wins fashion.

When the cluster starts up, each sensu-backend scans the stored replicator definitions and starts a replicator process for each replicator definition. Source clusters with more than one sensu-backend will cause redundant writes. This is harmless, but you should consider it when designing a replicated system.

***NOTE**: Create a replicator for each resource type you want to replicate. Replicating `namespace` resources will **not** replicate the resources that belong to those namespaces.*

Delete a replicator

When you delete a replicator, the replicator will issue delete events to the remote cluster for all of the keys in its prefix. It will not issue a delete of the entire key prefix (just in case the prefix is shared by keys that are local to the remote cluster).

Rather than altering an existing replicator's connection details, delete and recreate the replicator with the new connection details.

Replicator configuration

Etcd replicators are etcd key space replicators. Replicators contain configuration for forwarding a set of keys from one etcd cluster to another. Replicators are configured by specifying the TLS details of the remote cluster, its URL, and a resource type.

Etcd replicator specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API version of the etcd-replicators API. Always `federation/v1`.
required	true
type	String
example	**YML** `api_version: federation/v1` **JSON** `{ "api_version": "federation/v1" }`

metadata

description	Top-level scope that contains the replicator `name` and `created_by` value. Namespace is not supported in the metadata because etcd replicators are cluster-wide resources.
required	true
type	Map of key-value pairs
example	**YML** `metadata:` `  name: my_replicator` `  created_by: admin` **JSON** `{` `  "metadata": {` `  "name": "my_replicator",` `  "created_by": "admin"` `  }` `}`

spec

description	Top-level map that includes the replicator spec attributes.
required	true
type	Map of key-value pairs
example	**YML** `spec:` `  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem` `  cert: /path/to/ssl/cert.pem` `  key: /path/to/ssl/key.pem` `  insecure: false`

```
  url: http://127.0.0.1:2379
  api_version: core/v2
  resource: Role
  replication_interval_seconds: 30
```

JSON

```
{
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-
authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Always `EtcdReplicator.`
required	true
type	String
example	**YML** `type: EtcdReplicator` **JSON** `{ "type": "EtcdReplicator" }`

```
}
```

Metadata attributes

created_by	
description	Username of the Sensu user who created the replicator or last updated the replicator. Sensu automatically populates the `created_by` field when the replicator is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

name	
description	Replicator name used internally by Sensu.
required	true
type	String
example	**YML** `name: my_replicator`

JSON

```
{
  "name": "my_replicator"
}
```

Spec attributes

api_version	
description	Sensu API version of the resource to replicate.
required	false
type	String
default	`core/v2`
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

ca_cert	
description	Path to the PEM-format CA certificate to use for TLS client authentication.
required	true if `insecure: false` (the default configuration). If `insecure: true`, `ca_cert` is not required.

type	String

YML

example

```
ca_cert: /path/to/trusted-certificate-authorities.pem
```

JSON

```
{
  "ca_cert": "/path/to/trusted-certificate-authorities.pem"
}
```

cert

description	Path to the PEM-format certificate to use for TLS client authentication. This certificate is required for secure client communication.
required	true if `insecure: false` (the default configuration). If `insecure: true`, `cert` is not required.
type	String

YML

example

```
cert: /path/to/ssl/cert.pem
```

JSON

```
{
  "cert": "/path/to/ssl/cert.pem"
}
```

insecure

description	`true` to disable transport security. Otherwise, `false`.

***WARNING**: Disable transport security with care.*

required	false
type	Boolean
default	`false`
example	**YML** `insecure: false` **JSON** `{ "insecure": false }`

key

description	Path to the PEM-format key file associated with the `cert` to use for TLS client authentication. This key and its corresponding certificate are required for secure client communication.
required	true if `insecure: false` (the default configuration). If `insecure: true`, `key` is not required.
type	String
example	**YML** `key: /path/to/ssl/key.pem` **JSON** `{ "key": "/path/to/ssl/key.pem" }`

namespace

description	Namespace to constrain replication to. If you do not include `namespace`, all namespaces for a given resource are replicated.
required	false
type	String
example	**YML** `namespace: default` **JSON** `{ "namespace": "default" }`

replication_interval_seconds

description	Interval at which the resource will be replicated. In seconds.
required	false
type	Integer
default	30
example	**YML** `replication_interval_seconds: 30` **JSON** `{`

```
  "replication_interval_seconds": 30
}
```

resource

description	Name of the resource to replicate.
required	true
type	String
example	**YML** `resource: Role` **JSON** `{ "resource": "Role" }`

url

description	Destination cluster URL. If specifying more than one, use a comma to separate. Replace with a non-default value for secure client communication.
required	true
type	String
example	**YML** `url: http://127.0.0.1:2379` **JSON**

```
{
  "url": "http://127.0.0.1:2379"
}
```

Control Access

Sensu administrators control access by authentication and authorization.

Authentication verifies user identities to confirm that users are who they say they are. Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool. You can use Sensu's built-in basic authentication or configure external authentication providers.

> ***NOTE**: For API-specific authentication, read the API overview and Use API keys to authenticate to Sensu.*

Authorization establishes and manages user permissions: the extent of access users have for different Sensu resources. Configure authorization with role-based access control (RBAC) to exercise fine-grained control over how they interact with Sensu resources.

Authentication

Sensu web UI and sensuctl command line tool users can authenticate via built-in basic authentication or Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) when the administrator configures external single sign-on (SSO) authentication providers.

Sensu agents authenticate to the Sensu backend using either basic or mutual transport layer security (TLS) authentication.

Use built-in basic authentication

Sensu's built-in basic authentication allows you to create and manage user credentials (usernames and passwords) with core/v2/users API endpoints, either directly or using sensuctl. The basic authentication provider does not depend on external services and is not configurable.

Sensu hashes user passwords using the bcrypt algorithm and records the basic authentication credentials in etcd.

Use a single sign-on (SSO) authentication provider

***COMMERCIAL FEATURE**: Access authentication providers for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

In addition to built-in basic authentication, Sensu includes commercial support for single sign-on (SSO) authentication using external authentication providers via Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).

Read Configure single sign-on (SSO) authentication for general information about configuring an SSO authentication provider. Read the LDAP, AD, or OIDC reference documentation for provider-specific information.

Authorization

After you set up authentication, configure authorization via role-based access control (RBAC) to give those users permissions within Sensu. RBAC allows you to specify actions users are allowed to take against resources, within namespaces or across all namespaces, based on roles bound to the user or to one or more groups the user is a member of. Read Create a read-only user for an example.

- **Namespaces** partition resources within Sensu. Sensu entities, checks, handlers, and other namespaced resources belong to a single namespace.
- **Roles** create sets of permissions (like GET and DELETE) tied to resource types. **Cluster roles** apply permissions across all namespaces and may include access to cluster-wide resources like users and namespaces.
- **Role bindings** assign a role to a set of users and groups within a namespace. **Cluster role bindings** assign a cluster role to a set of users and groups across all namespaces.

To enable permissions for external users and groups within Sensu, you can create a set of roles, cluster roles, role bindings, and cluster role bindings that map to the usernames and group names in your authentication provider.

After you configure an authentication provider and establish the roles and bindings to grant authenticated users the desired privileges, those users can log in via sensuctl and the web UI using a single-sign-on username and password. Users do *not* need to provide the username prefix for the authentication provider when logging in to Sensu.

Configure single sign-on (SSO) authentication

***COMMERCIAL FEATURE**: Access authentication providers for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool.

In addition to the built-in basic authentication, Sensu offers commercial support for using Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) for single sign-on (SSO) authentication.

This guide describes general information for configuring an authentication provider for SSO. Read the LDAP, AD, or OIDC reference documentation for provider-specific examples and specifications.

Configure authentication providers

To configure an external authentication provider for SSO, first write an authentication provider configuration definition. Follow the examples and specifications for your provider:

- **Lightweight Directory Access Protocol (LDAP)**, including standards-compliant tools like OpenLDAP (configuration examples and specification)
- **Microsoft Active Directory (AD)**, including Azure AD (configuration examples and specification)
- **OpenID Connect 1.0 protocol (OIDC)**, including tools like Okta and PingFederate (configuration examples and specification)

Save your configuration definition to a file, such as `authconfig.yaml` or `authconfig.json`.

After you have a saved configuration definition, you can apply the configuration with sensuctl. Log in to sensuctl as the default admin user and use sensuctl to apply your authentication provider configuration to Sensu:

SHELL

```
sensuctl create --file authconfig.yml
```

SHELL

```
sensuctl create --file authconfig.json
```

Use sensuctl to verify that your provider configuration was applied successfully:

```
sensuctl auth list
```

The response will list your authentication provider types and names:

```
  Type    Name
______ __________

ldap   openldap
```

Manage authentication providers

View and delete authentication providers with enterprise/authentication/v2 API endpoints or these sensuctl commands.

To view active authentication providers:

```
sensuctl auth list
```

To view configuration details for an authentication provider named `openldap`:

```
sensuctl auth info openldap
```

To delete an authentication provider named `openldap`:

```
sensuctl auth delete openldap
```

Use API keys to authenticate to Sensu

The Sensu API key feature (core/v2.APIKey) is a persistent universally unique identifier (UUID) that maps to a stored Sensu username. The advantages of authenticating with API keys rather than access tokens include:

- **More efficient integration**: Check and handler plugins and other code can integrate with the Sensu API without implementing the logic required to authenticate via the `/auth` API endpoint to periodically refresh the access token
- **Improved security**: API keys do not require providing a username and password in check or handler definitions
- **Better admin control**: API keys can be created and revoked without changing the underlying user's password…but keep in mind that API keys will continue to work even if the user's password changes

API keys are cluster-wide resources, so only cluster admins can grant, view, and revoke them.

> ***NOTE**: API keys are not supported for authentication providers such as LDAP and OIDC.*

API key authentication

Similar to the `Bearer [token]` Authorization header, `Key [api-key]` will be accepted as an Authorization header for authentication.

For example, a JWT `Bearer [token]` Authorization header might be:

```
curl -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

If you're using `Key [api-key]` to authenticate instead, the Authorization header might be:

```
curl -H "Authorization: Key $SENSU_API_KEY"
```

```
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

Here's an example request that uses API key authentication:

```
curl -H "Authorization: Key 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK` and the definitions for the checks in the default namespace.

Sensuctl management commands

> ***NOTE**: The API key resource is intentionally not compatible with `sensuctl create`.*

To use sensuctl to generate a new API key for the admin user, run:

```
sensuctl api-key grant admin
```

The response will include the new API key:

```
Created: /api/core/v2/apikeys/7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2
```

To bypass username/password authentication for sensuctl, add the `--api-key` global flag to specify your API key with sensuctl commands. For example:

```
sensuctl --api-key 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 event list
```

To get information about an API key:

SHELL

```
```

```
sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format yaml
```

SHELL

```
sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format wrapped-json
```

SHELL

```
sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format json
```

The response will include information about the API key in the specified format:

YML

```
---
type: APIKey
api_version: core/v2
metadata:
  created_by: admin
  name: 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2
spec:
  created_at: 1570718917
  username: admin
```

SHELL

```
{
  "type": "APIKey",
  "api_version": "core/v2",
  "metadata": {
    "name": "7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2",
    "created_by": "admin"
  },
  "spec": {
    "created_at": 1570718917,
    "username": "admin"
  }
}
```

JSON

```
{
  "metadata": {
    "name": "7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2",
    "created_by": "admin"
  },
  "username": "admin",
  "created_at": 1570718917
}
```

To get a list of all API keys:

```
sensuctl api-key list
```

The response lists all API keys along with the name of the user who created each key and the date and time each key was created:

```
             Name                Username        Created At
 ──────────────────────────────────────── ──────────── ──────────────────────────────────
  7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2   admin      2019-10-10 14:48:37 -0700 PDT
```

To revoke an API key for the admin user:

```
sensuctl api-key revoke 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --skip-confirm
```

The response will confirm that the API key is deleted:

```
Deleted
```

Create a read-only user with role-based access control

Role-based access control (RBAC) allows you to exercise fine-grained control over how Sensu users interact with Sensu resources. Use RBAC rules to achieve **multitenancy** so different projects and teams can share a Sensu instance.

Sensu RBAC helps different teams and projects share a Sensu instance. RBAC allows you to manage users and their access to resources based on **namespaces**, **groups**, **roles**, and **bindings**.

By default, Sensu includes a `default` namespace and an `admin` user with full permissions to create, modify, and delete resources within Sensu, including RBAC resources like users and roles. This guide requires a running Sensu backend and a sensuctl instance configured to connect to the backend as an `admin` user.

Create a read-only user

In this section, you'll create a user and assign them read-only access to resources within the `default` namespace using a **role** and a **role binding**.

1. Create a user with the username `alice` and assign them to the group `ops`:

   ```
   sensuctl user create alice --password='password' --groups=ops
   ```

   This command creates the following user:

   TEXT

   ```
   username: alice
   groups:
   - ops
   disabled: false
   ```

   TEXT

```
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

2. Create a `read-only` role with `get` and `list` permissions for all resources (`*`) within the `default` namespace:

```
sensuctl role create read-only --verb=get,list --resource=* --
namespace=default
```

This command creates the following role resource definition:

TEXT

```
---
type: Role
api_version: core/v2
metadata:
  name: read-only
spec:
  rules:
  - resource_names: null
    resources:
    - '*'
    verbs:
    - get
    - list
```

TEXT

```
{
  "type": "Role",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "read-only"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "*"
        ],
        "verbs": [
          "get",
          "list"
        ]
      }
    ]
  }
}
```

3. Create an `ops-read-only` role binding to assign the `read-only` role to the `ops` group:

```
sensuctl role-binding create ops-read-only --role=read-only --group=ops
```

This command creates the following role binding resource definition:

TEXT

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: ops-read-only
spec:
  role_ref:
    name: read-only
    type: Role
  subjects:
  - name: ops
    type: Group
```

TEXT

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-read-only"
  },
  "spec": {
    "role_ref": {
      "name": "read-only",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

All users in the `ops` group now have read-only access to all resources within the default namespace. You can also use role bindings to tie roles directly to users using the `--user` flag.

To manage your RBAC configuration, use the `sensuctl user`, `sensuctl role`, and `sensuctl role-binding` commands.

Create a cluster-wide event-reader user

Suppose you want to create a user with read-only access to events across all namespaces.Because you want this role to have cluster-wide permissions, you'll need to create a **cluster role** and a **cluster role binding**.

1. Create a user with the username `bob` and assign them to the group `ops`:

```
sensuctl user create bob --password='password' --groups=ops
```

This command creates the following user:

TEXT

```
username: bob
groups:
- ops
disabled: false
```

TEXT

```
{
  "username": "bob",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

2. Create a `global-event-reader` cluster role with `get` and `list` permissions for `events` across all namespaces:

```
sensuctl cluster-role create global-event-reader --verb=get,list --
resource=events
```

This command creates the following cluster role resource definition:

TEXT

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: global-event-reader
spec:
  rules:
```

```
  - resource_names: null
    resources:
    - events
    verbs:
    - get
    - list
```

TEXT

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "global-event-reader"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "events"
        ],
        "verbs": [
          "get",
          "list"
        ]
      }
    ]
  }
}
```

3. Create an `ops-event-reader` cluster role binding to assign the `global-event-reader` role to the `ops` group:

```
sensuctl cluster-role-binding create ops-event-reader --cluster-role=global-event-reader --group=ops
```

This command creates the following cluster role binding resource definition:

TEXT

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: ops-event-reader
spec:
  role_ref:
    name: global-event-reader
    type: ClusterRole
  subjects:
  - name: ops
    type: Group
```

TEXT

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-event-reader"
  },
  "spec": {
    "role_ref": {
      "name": "global-event-reader",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

All users in the `ops` group now have read-only access to events across all namespaces.

Next steps

Now that you know how to create a user, a role, and a role binding to assign a role to a user, check out the RBAC reference for in-depth documentation on role-based access control, examples, and information about cluster-wide permissions.

Read about monitoring as code with Sensu and learn how to use SensuFlow to synchronize your monitoring and observability code with your Sensu deployments.

Create limited service accounts

In some cases, you may want to allow an application or service to interact with Sensu resources. Use Sensu's role-based access control (RBAC) to create and configure accounts that represent applications or services rather than individual human users. These limited service accounts give you fine-grained control of the access and permissions the application or service needs.

For example, you might develop a service that displays a high-level view of your webserver statuses based on an aggregate check. The service itself only needs an API key and permission to read the results of checks executed on your webservers so it can route the check results to the status display. No human user needs to log into the service, and the service does not need edit or delete permissions. A limited service account can provide only the necessary access and permissions.

Limited service accounts are also useful for performing automated processes. This guide explains how to create a limited service account to use with the sensu/sensu-ec2-handler dynamic runtime asset to automatically remove AWS EC2 instances that are not in a pending or running state.

By default, Sensu includes a `default` namespace and an `admin` user with full permissions to create, modify, and delete resources within Sensu, including the RBAC resources required to configure a limited service account. This guide requires a running Sensu backend and a sensuctl instance configured to connect to the backend as the `admin` user.

Create a limited service account

A limited service account requires:

- A user.
- A role with get, list, and delete permissions for resources within the `default` namespace.
- A role binding that ties the role to the user.
- An API key for the user.

***NOTE**: To use a service account to manage Sensu resources in more than one namespace, create a cluster role instead of a role and a cluster role binding instead of a role binding.*

1. Create a user with the username `ec2-service` and a dynamically created random password:

```
sensuctl user create ec2-service --password=$(head -c1M /dev/urandom |
sha512sum | cut -d' ' -f1 | head -c 32)
```

This command creates the following user definition:

TEXT

```
---
type: User
api_version: core/v2
metadata:
  name: ec2-service
spec:
  disabled: false
  username: ec2-service
```

TEXT

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {
    "name": "ec2-service"
  },
  "spec": {
    "disabled": false,
    "username": "ec2-service"
  }
}
```

2. Create a `ec2-delete` role with get, list, and delete permissions for entity resources within the `default` namespace:

```
sensuctl role create ec2-delete --verb get,list,delete --resource entities --
namespace default
```

This command creates the role that has the permissions your service account will need:

TEXT

```
---
type: Role
api_version: core/v2
metadata:
  name: ec2-delete
spec:
  rules:
  - resource_names: null
    resources:
    - entities
    verbs:
    - get
    - list
    - delete
```

TEXT

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "ec2-delete"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "entities"
        ],
        "verbs": [
          "get",
          "list",
          "delete"
        ]
      }
    ]
```

```
  }
}
```

3. Create an `ec2-service-delete` role binding to assign the `ec2-delete` role to the `ec2-service` user:

```
sensuctl role-binding create ec2-service-delete --role ec2-delete --user ec2-service
```

This command creates the role binding that ties the correct permissions (via the `ec2-delete` role) with your service account (via the user `ec2-service`):

TEXT

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: ec2-service-delete
spec:
  role_ref:
    name: ec2-delete
    type: Role
  subjects:
  - name: ec2-service
    type: User
```

TEXT

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ec2-service-delete"
  },
  "spec": {
    "role_ref": {
      "name": "ec2-delete",
```

```
        "type": "Role"
      },
      "subjects": [
        {
          "name": "ec2-service",
          "type": "User"
        }
      ]
    }
  }
```

4. Create an API key for the `ec2-service` user:

   ```
   sensuctl api-key grant ec2-service
   ```

   The response will include an API key that is assigned to the `ec2-service` user, which you will need to configure the EC2 handler.

The `ec2-service` limited service account is now ready to use with the sensu/sensu-ec2-handler dynamic runtime asset.

Add the sensu/sensu-ec2-handler dynamic runtime asset

To power the handler to remove AWS EC2 instances, use sensuctl to add the sensu/sensu-ec2-handler dynamic runtime asset:

```
sensuctl asset add sensu/sensu-ec2-handler:0.4.0
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-ec2-handler:0.4.0
added asset: sensu/sensu-ec2-handler:0.4.0

You have successfully added the Sensu asset resource, but the asset will not get
```

```
  downloaded until
  it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
  appropriate
  resource, populate the "runtime_assets" field with ["sensu/sensu-ec2-handler"].
```

You can also download the dynamic runtime asset definition from Bonsai and register the asset with `sensuctl create --file filename.yml`.

> ***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read the asset reference for more information about dynamic runtime asset builds.*

Configure an EC2 handler for the service account

To configure the EC2 handler, you will need AWS account credentials and details for the AWS instance you want to manage, like the AWS instance ID. You will also need the API key for the `ec2-service` user.

> ***NOTE**: Use secrets management to configure environment variables for your AWS access and secret keys and the `ec2-service` user's API key. Do not expose this sensitive information by listing it directly in the handler definition.*
>
> *The Sensu Go EC2 Handler's Bonsai page includes an example for configuring secrets definitions with Sensu's `Env` secrets provider.*

In the following code, replace these bracketed placeholders with valid values:

- `<AWS_REGION>`: the AWS region where your EC2 instance is located.
- `<AWS_INSTANCE_ID_LABEL>`: the Sensu entity label that contains the AWS instance ID. If your AWS EC2 instance entities do not include labels that specify the instance ID, use the `aws-instance-id` attribute instead and enter the AWS instance ID itself as the value.
- `<http://localhost:8080>`: the Sensu API URL.

You can also adjust the `aws-allowed-instance-states` value to include any of the Sensu EC2 integration's available states. This example lists only "pending" and "running."

Then, run this command with your valid values in place to create the handler defintion:

SHELL

```
cat << EOF | sensuctl create
---
type: Handler
api_version: core/v2
metadata:
  name: sensu-ec2-handler
spec:
  type: pipe
  runtime_assets:
    - sensu/sensu-ec2-handler
  filters:
    - is_incident
    - not_silenced
  command: >-
    sensu-ec2-handler
    --aws-region <AWS_REGION>
    --aws-instance-id-label <AWS_INSTANCE_ID_LABEL>
    --aws-allowed-instance-states pending,running
    --sensu-api-url <http://localhost:8080>
  secrets:
    - name: AWS_ACCESS_KEY_ID
      secret: <YOUR_AWS_ACCESS_KEY_ID>
    - name: AWS_SECRET_KEY
      secret: <YOUR_AWS_SECRET_KEY>
    - name: SENSU_API_KEY
      secret: <YOUR_SENSU_API_KEY>
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-ec2-handler"
  },
  "spec": {
```

```
    "type": "pipe",
    "runtime_assets": [
      "sensu/sensu-ec2-handler"
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "command": "sensu-ec2-handler --aws-region <AWS_REGION> --aws-instance-id-label
AWS_INSTANCE_ID_LABEL --aws-allowed-instance-states pending,running --sensu-api-url
<http://localhost:8080">,
    "secrets": [
      {
        "name": "AWS_ACCESS_KEY_ID",
        "secret": "<YOUR_AWS_ACCESS_KEY_ID>"
      },
      {
        "name": "AWS_SECRET_KEY",
        "secret": "<YOUR_AWS_SECRET_KEY>"
      },
      {
        "name": "SENSU_API_KEY",
        "secret": "<YOUR_SENSU_API_KEY>"
      }
    ]
  }
}
EOF
```

The handler will use the provided AWS credentials to check the specified EC2 instance. If the instance's status is not "pending" or "running," the handler will use the `ec2-service` user's API key to remove the corresponding entity.

***NOTE**: Instead of directly referencing your `AWS_ACCESS_KEY_ID`, `AWS_SECRET_KEY`, and `SENSU_API_KEY` as shown in the `sensu-ec2-handler` example handler definition above, use secrets management to configure these values as environment variables.*

Best practices for limited service accounts

Follow these best practices for creating and managing limited service accounts:

- Use unique and specific names for limited service accounts. Names should identify the accounts as limited service accounts as well as the associated services.
- Restrict limited service account access to only the namespaces and role permissions they need to operate properly. Adjust namespaces and permissions if needed by updating the role or cluster role that is tied to the service account.
- Promptly delete unused limited service accounts to make sure they do not become security risks.

Active Directory (AD) reference

***COMMERCIAL FEATURE**: Access active directory (AD) authentication for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool.

In addition to the built-in basic authentication, Sensu offers commercial support for using Microsoft Active Directory (AD) for single sign-on (SSO) authentication. The AD authentication provider is based on the LDAP authentication provider.

To use AD authentication for Azure, follow Microsoft's tutorial to set up secure LDAP in your Azure account and create the host and certificates you need.

For general information about configuring authentication providers, read Configure single sign-on (SSO) authentication.

AD configuration examples

Example AD configuration: Minimum required attributes

YML

```
---
type: ad
api_version: authentication/v2
metadata:
  name: activedirectory
spec:
  servers:
  - group_search:
      base_dn: dc=acme,dc=org
    host: 127.0.0.1
    user_search:
```

```
      base_dn: dc=acme,dc=org
```

JSON

```
{
  "type": "ad",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "group_search": {
          "base_dn": "dc=acme,dc=org"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "activedirectory"
  }
}
```

Example AD configuration: All attributes

YML

```
---
type: ad
api_version: authentication/v2
metadata:
  name: activedirectory
spec:
  allowed_groups: []
  groups_prefix: ad
  servers:
  - binding:
      password: YOUR_PASSWORD
```

```
      user_dn: cn=binder,cn=users,dc=acme,dc=org
    client_cert_file: /path/to/ssl/cert.pem
    client_key_file: /path/to/ssl/key.pem
    default_upn_domain: example.org
    include_nested_groups: true
    group_search:
      attribute: member
      base_dn: dc=acme,dc=org
      name_attribute: cn
      object_class: group
    host: 127.0.0.1
    insecure: false
    port: 636
    security: tls
    trusted_ca_file: /path/to/trusted-certificate-authorities.pem
    user_search:
      attribute: sAMAccountName
      base_dn: dc=acme,dc=org
      name_attribute: displayName
      object_class: person
  username_prefix: ad
```

JSON

```
{
  "type": "ad",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "default_upn_domain": "example.org",
        "include_nested_groups": true,
        "binding": {
          "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
```

```
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "member",
          "name_attribute": "cn",
          "object_class": "group"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "sAMAccountName",
          "name_attribute": "displayName",
          "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ad",
    "username_prefix": "ad"
  },
  "metadata": {
    "name": "activedirectory"
  }
}
```

Example AD configuration: Use `memberOf` attribute instead of `group_search`

AD automatically returns a `memberOf` attribute in users' accounts. The `memberOf` attribute contains the user's group membership, which effectively removes the requirement to look up the user's groups.

To use the `memberOf` attribute in your AD implementation, remove the `group_search` object from your AD config:

YML

```
---
type: ad
api_version: authentication/v2
metadata:
  name: activedirectory
spec:
  servers:
```

```
    host: 127.0.0.1
    user_search:
      base_dn: dc=acme,dc=org
```

JSON

```
{
  "type": "ad",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "activedirectory"
  }
}
```

After you configure AD to use the `memberOf` attribute, the `debug` log level will include the following log entries:

```
{"component":"authentication/v2","level":"debug","msg":"using the \"memberOf\"
attribute to determine the group membership of user \"user1\"","time":"2020-06-
25T14:10:58-04:00"}
{"component":"authentication/v2","level":"debug","msg":"found 1 LDAP group(s):
[\"sensu\"]","time":"2020-06-25T14:10:58-04:00"}
```

AD specification

AD top-level attributes

type

description	Top-level attribute that specifies the `sensuctl create` resource type. For AD definitions, the `type` should always be `ad`.
required	true
type	String
example	**YML** `type: ad` **JSON** `{ "type": "ad" }`

api_version

description	Top-level attribute that specifies the Sensu API group and version. For AD definitions, the `api_version` should always be `authentication/v2`.
required	true
type	String
example	**YML** `api_version: authentication/v2` **JSON** `{ "api_version": "authentication/v2" }`

```
}
```

metadata

description	Top-level map that contains the AD definition `name`. Review the metadata attributes for details.
required	true
type	Map of key-value pairs
example	**YML** ```yaml metadata: name: activedirectory ``` **JSON** ```json { "metadata": { "name": "activedirectory" } } ```

spec

description	Top-level map that includes the AD spec attributes.
required	true
type	Map of key-value pairs
example	**YML** ```yaml spec: servers: - host: 127.0.0.1 ```

```
    port: 636
    insecure: false
    security: tls
    trusted_ca_file: "/path/to/trusted-certificate-
authorities.pem"
    client_cert_file: "/path/to/ssl/cert.pem"
    client_key_file: "/path/to/ssl/key.pem"
    default_upn_domain: example.org
    include_nested_groups: true
    binding:
      user_dn: cn=binder,cn=users,dc=acme,dc=org
      password: YOUR_PASSWORD
    group_search:
      base_dn: dc=acme,dc=org
      attribute: member
      name_attribute: cn
      object_class: group
    user_search:
      base_dn: dc=acme,dc=org
      attribute: sAMAccountName
      name_attribute: displayName
      object_class: person
  allowed_groups: []
  groups_prefix: ad
  username_prefix: ad
```

JSON

```
{
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "default_upn_domain": "example.org",
        "include_nested_groups": true,
```

```
        "binding": {
          "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "member",
          "name_attribute": "cn",
          "object_class": "group"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "sAMAccountName",
          "name_attribute": "displayName",
          "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ad",
    "username_prefix": "ad"
  }
}
```

AD metadata attributes

name	
description	A unique string used to identify the AD configuration. Names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`).
required	true
type	String
example	**YML** `name: activedirectory`

JSON

```
{
  "name": "activedirectory"
}
```

AD spec attributes

servers	
description	The list of AD servers to use. During the authentication process, Sensu attempts to authenticate against each AD server in sequence until authentication is successful or there are no more servers to try.
required	true
type	Array
example	**YML** `servers:` `- host: 127.0.0.1` `  port: 636` `  insecure: false` `  security: tls` `  trusted_ca_file: "/path/to/trusted-certificate-authorities.pem"` `  client_cert_file: "/path/to/ssl/cert.pem"` `  client_key_file: "/path/to/ssl/key.pem"` `  default_upn_domain: example.org` `  include_nested_groups: true` `  binding:` `  user_dn: cn=binder,cn=users,dc=acme,dc=org` `  password: YOUR_PASSWORD` `  group_search:` `  base_dn: dc=acme,dc=org` `  attribute: member`

```
    name_attribute: cn
    object_class: group
  user_search:
    base_dn: dc=acme,dc=org
    attribute: sAMAccountName
    name_attribute: displayName
    object_class: person
```

JSON

```
{
  "servers": [
    {
      "host": "127.0.0.1",
      "port": 636,
      "insecure": false,
      "security": "tls",
      "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem",
      "client_cert_file": "/path/to/ssl/cert.pem",
      "client_key_file": "/path/to/ssl/key.pem",
      "default_upn_domain": "example.org",
      "include_nested_groups": true,
      "binding": {
        "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
        "password": "YOUR_PASSWORD"
      },
      "group_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "member",
        "name_attribute": "cn",
        "object_class": "group"
      },
      "user_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "sAMAccountName",
        "name_attribute": "displayName",
        "object_class": "person"
      }
    }
  ]
}
```

allowed_groups

description	An array of allowed AD group strings to include in the tokenized identity claim. Use to specify which groups to encode in the authentication provider's JSON Web Token (JWT) when the authenticated AD user is a member of many groups and the tokenized identity claim would be too large for correct web client operation.
required	false
type	Array
example	**YML** `allowed_groups:` `- sensu-viewers` `- sensu-operators` **JSON** `{` `"allowed_groups": [` `"sensu-viewers",` `"sensu-operators"` `]` `}`

groups_prefix

description	The prefix added to all AD groups. Sensu appends the groups_prefix with a colon. For example, for the groups_prefix `ad` and the group `dev`, the resulting group name in Sensu is `ad:dev`. Use the groups_prefix when integrating AD groups with Sensu RBAC role bindings and cluster role bindings.

required	false
type	String

YML

example

```
groups_prefix: ad
```

JSON

```
{
  "groups_prefix": "ad"
}
```

username_prefix

description	The prefix added to all AD usernames. Sensu appends the username_prefix with a colon. For example, for the username_prefix `ad` and the user `alice`, the resulting username in Sensu is `ad:alice`. Use the username_prefix when integrating AD users with Sensu RBAC role bindings and cluster role bindings. Users *do not* need to provide the username_prefix when logging in to Sensu.
required	false
type	String

YML

example

```
username_prefix: ad
```

JSON

```
{
  "username_prefix": "ad"
}
```

AD server attributes

host

description	AD server IP address or fully qualified domain name (FQDN).
required	true
type	String
example	**YML** `host: 127.0.0.1` **JSON** `{ "host": "127.0.0.1" }`

port

description	AD server port.
required	true
type	Integer
default	`389` for insecure connections; `636` for TLS connections
example	**YML** `port: 636` **JSON** `{ "port": 636 }`

insecure

description	Skips SSL certificate verification when set to `true`. ***WARNING**: Do not use an insecure connection in production environments.*
required	false
type	Boolean
default	`false`
example	**YML** ```yaml insecure: false ``` **JSON** ```json { "insecure": false } ```

security

description	Determines the encryption type to be used for the connection to the AD server: `insecure` (unencrypted connection; not recommended for production), `tls` (secure encrypted connection), or `starttls` (unencrypted connection upgrades to a secure connection).
type	String
default	`tls` **YML**

example	```yaml security: tls ``` **JSON** ```json { "security": "tls" } ```

trusted_ca_file

description	Path to an alternative CA bundle file in PEM format to be used instead of the system's default bundle. This CA bundle is used to verify the server's certificate.
required	false
type	String
example	**YML** ```yaml trusted_ca_file: /path/to/trusted-certificate-authorities.pem ``` **JSON** ```json { "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem" } ```

client_cert_file

description	Path to the certificate that should be sent to the server if requested.
required	false

type	String
example	**YML** `client_cert_file: /path/to/ssl/cert.pem` **JSON** `{ "client_cert_file": "/path/to/ssl/cert.pem" }`

client_key_file

description	Path to the key file associated with the `client_cert_file`.
required	false
type	String
example	**YML** `client_key_file: /path/to/ssl/key.pem` **JSON** `{ "client_key_file": "/path/to/ssl/key.pem" }`

binding

description	The AD account that performs user and group lookups. If your server supports anonymous binding, you can omit the `user_dn` or `password` attributes to query the directory without credentials. To use anonymous binding with AD, the `ANONYMOUS LOGON` object requires read

permissions for users and groups. Review the binding attributes for details.

required	false
type	Map

YML

```
binding:
  user_dn: cn=binder,cn=users,dc=acme,dc=org
  password: YOUR_PASSWORD
```

JSON

```
{
  "binding": {
    "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
    "password": "YOUR_PASSWORD"
  }
}
```

group_search

description	Search configuration for groups. Review the group search attributes for more information. Remove the `group_search` object from your configuration to use the `memberOf` attribute instead.
required	false
type	Map

YML

```
group_search:
  base_dn: dc=acme,dc=org
  attribute: member
  name_attribute: cn
  object_class: group
```

JSON

```
{
  "group_search": {
    "base_dn": "dc=acme,dc=org",
    "attribute": "member",
    "name_attribute": "cn",
    "object_class": "group"
  }
}
```

user_search

description	Search configuration for users. Review the user search attributes for more information.
required	true
type	Map

YML

example

```
user_search:
  base_dn: dc=acme,dc=org
  attribute: sAMAccountName
  name_attribute: displayName
  object_class: person
```

JSON

```
{
  "user_search": {
    "base_dn": "dc=acme,dc=org",
    "attribute": "sAMAccountName",
    "name_attribute": "displayName",
    "object_class": "person"
  }
}
```

default_upn_domain

description

Enables UPN authentication when set. The default UPN suffix that will be appended to the username when a domain is not specified during login (for example, `user` becomes `user@defaultdomain.xyz`).

> ***WARNING**: When using UPN authentication, users must re-authenticate to apply any changes to group membership on the AD server since their last authentication. For example, if you remove a user from a group with administrator permissions for the current session (such as a terminated employee), Sensu will not apply the change until the user logs out and tries to start a new session. Likewise, under UPN, users cannot be forced to log out of Sensu. To apply group membership updates without re-authentication, specify a binding account or enable anonymous binding.*

required	false
type	String

YML

```
default_upn_domain: example.org
```

JSON

```
{
  "default_upn_domain": "example.org"
}
```

include_nested_groups

description

If `true`, the group search includes any nested groups a user is a member of. If `false`, the group search includes only the top-level groups a user is a member of.

required	false
type	Boolean

YML

example

```
include_nested_groups: true
```

JSON

```
{
  "include_nested_groups": true
}
```

AD binding attributes

user_dn

description	The AD account that performs user and group lookups. We recommend using a read-only account. Use the distinguished name (DN) format, such as `cn=binder,cn=users,dc=domain,dc=tld`. If your server supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String

YML

example

```
user_dn: cn=binder,cn=users,dc=acme,dc=org
```

JSON

```
{
  "user_dn": "cn=binder,cn=users,dc=acme,dc=org"
}
```

password	
description	Password for the `user_dn` account. If your server supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String
example	**YML** `password: YOUR_PASSWORD` **JSON** `{ "password": "YOUR_PASSWORD" }`

AD group search attributes

base_dn	
description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String
example	**YML** `base_dn: dc=acme,dc=org` **JSON**

```
{
  "base_dn": "dc=acme,dc=org"
}
```

attribute

description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false
type	String
default	`member`

YML

example

```
attribute: member
```

JSON

```
{
  "attribute": "member"
}
```

name_attribute

description	Represents the attribute to use as the entry name.
required	false
type	String
default	`cn`

YML

example

```
name_attribute: cn
```

JSON

```
{
  "name_attribute": "cn"
}
```

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`group`

YML

example

```
object_class: group
```

JSON

```
{
  "object_class": "group"
}
```

AD user search attributes

base_dn

description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String
example	**YML** `base_dn: dc=acme,dc=org` **JSON** `{ "base_dn": "dc=acme,dc=org" }`

attribute

description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false
type	String
default	`sAMAccountName`
example	**YML** `attribute: sAMAccountName` **JSON** `{ "attribute": "sAMAccountName" }`

name_attribute

description	Represents the attribute to use as the entry name.
required	false
type	String
default	`displayName`
example	**YML** `name_attribute: displayName` **JSON** `{ "name_attribute": "displayName" }`

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`person`
example	**YML** `object_class: person` **JSON** `{ "object_class": "person" }`

```
}
```

AD troubleshooting

To troubleshoot any issue with AD authentication, start by increasing the log verbosity of sensu-backend to the debug log level. Most authentication and authorization errors are only displayed on the debug log level to avoid flooding the log files.

> ***NOTE**: If you can't locate any log entries referencing AD authentication, run sensuctl auth list to make sure that you successfully installed the AD provider.*

Authentication

This section lists common authentication error messages and describes possible solutions for each of them.

`failed to connect: AD Result Code 200 "Network Error"`

The AD provider couldn't establish a TCP connection to the AD server. Verify the `host` and `port` attributes. If you are not using AD over TLS/SSL, make sure to set the value of the `security` attribute to `insecure` for plaintext communication.

`certificate signed by unknown authority`

If you are using a self-signed certificate, make sure to set the `insecure` attribute to `true`. This will bypass verification of the certificate's signing authority.

`failed to bind: ...`

The first step for authenticating a user with the AD provider is to bind to the AD server using the service account specified in the `binding` object. Make sure the `user_dn` attribute specifies a valid **DN** and that its password is correct.

`user <username> was not found`

The user search failed. No user account could be found with the given username. Check the `user_search` object and make sure that:

- The specified `base_dn` contains the requested user entry DN
- The specified `attribute` contains the *username* as its value in the user entry
- The `object_class` attribute corresponds to the user entry object class

`ad search for user <username> returned x results, expected only 1`

The user search returned more than one user entry, so the provider could not determine which of these entries to use. Change the `user_search` object so the provided `username` can be used to uniquely identify a user entry. Here are two methods to try:

- Adjust the `attribute` so its value (which corresponds to the `username`) is unique among the user entries
- Adjust the `base_dn` so it only includes one of the user entries

`ad entry <DN> missing required attribute <name_attribute>`

The user entry returned (identified by `<DN>`) doesn't include the attribute specified by `name_attribute` object, so the AD provider could not determine which attribute to use as the username in the user entry. Adjust the `name_attribute` so it specifies a human-readable name for the user.

`ad group entry <DN> missing <name_attribute> and cn attributes`

The group search returned a group entry (identified by `<DN>`) that doesn't have the `name_attribute` object or a `cn` attribute, so the AD provider could not determine which attribute to use as the group name in the group entry. Adjust the `name_attribute` so it specifies a human-readable name for the group.

Authorization

Once authenticated, each user needs to be granted permissions via either a `ClusterRoleBinding` or a `RoleBinding`.

The way AD users and AD groups can be referred as subjects of a cluster role or role binding depends on the `groups_prefix` and `username_prefix` configuration attributes values of the AD provider. For example, for the groups_prefix `ad` and the group `dev`, the resulting group name in Sensu is `ad:dev`.

Permissions are not granted via the AD group(s)

During authentication, the AD provider will print all groups found in AD (for example, `found 1 group(s): [dev]`) in the logs. Keep in mind that this group name does not contain the `groups_prefix` at this point.

The Sensu backend logs each attempt made to authorize an RBAC request. This is useful for determining why a specific binding didn't grant the request. For example:

```
[...] the user is not a subject of the ClusterRoleBinding cluster-admin [...]
[...] could not authorize the request with the ClusterRoleBinding system:user [...]
[...] could not authorize the request with any ClusterRoleBindings [...]
```

Lightweight Directory Access Protocol (LDAP) reference

***COMMERCIAL FEATURE**: Access Lightweight Directory Access Protocol (LDAP) authentication for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool.

In addition to the built-in basic authentication, Sensu offers commercial support for a standards-compliant Lightweight Directory Access Protocol (LDAP) tool for single sign-on (SSO) authentication. The Sensu LDAP authentication provider is tested with OpenLDAP. If you're using AD, head to the AD section.

For general information about configuring authentication providers, read Configure single sign-on (SSO) authentication.

LDAP configuration examples

Example LDAP configuration: Minimum required attributes

YML

```
---
type: ldap
api_version: authentication/v2
metadata:
  name: openldap
spec:
  servers:
  - group_search:
      base_dn: dc=acme,dc=org
    host: 127.0.0.1
    user_search:
```

```
      base_dn: dc=acme,dc=org
```

JSON

```
{
  "type": "ldap",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "group_search": {
          "base_dn": "dc=acme,dc=org"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "openldap"
  }
}
```

Example LDAP configuration: All attributes

YML

```
---
type: ldap
api_version: authentication/v2
metadata:
  name: openldap
spec:
  allowed_groups: []
  groups_prefix: ldap
  servers:
  - binding:
      password: YOUR_PASSWORD
```

```
      user_dn: cn=binder,dc=acme,dc=org
    client_cert_file: /path/to/ssl/cert.pem
    client_key_file: /path/to/ssl/key.pem
    group_search:
      attribute: member
      base_dn: dc=acme,dc=org
      name_attribute: cn
      object_class: groupOfNames
    host: 127.0.0.1
    insecure: false
    port: 636
    security: tls
    trusted_ca_file: /path/to/trusted-certificate-authorities.pem
    user_search:
      attribute: uid
      base_dn: dc=acme,dc=org
      name_attribute: cn
      object_class: person
  username_prefix: ldap
```

JSON

```
{
  "type": "ldap",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "binding": {
          "user_dn": "cn=binder,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "member",
```

```
          "name_attribute": "cn",
          "object_class": "groupOfNames"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "uid",
          "name_attribute": "cn",
          "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ldap",
    "username_prefix": "ldap"
  },
  "metadata": {
    "name": "openldap"
  }
}
```

Example LDAP configuration: Use `memberOf` attribute instead of `group_search`

If your LDAP server is configured to return a `memberOf` attribute when you perform a query, you can use `memberOf` in your Sensu LDAP implementation instead of `group_search`. The `memberOf` attribute contains the user's group membership, which effectively removes the requirement to look up the user's groups.

To use the `memberOf` attribute in your LDAP implementation, remove the `group_search` object from your LDAP config:

YML

```
---
type: ldap
api_version: authentication/v2
metadata:
  name: openldap
spec:
  servers:
    host: 127.0.0.1
    user_search:
```

```
      base_dn: dc=acme,dc=org
```

JSON

```
{
  "type": "ldap",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "openldap"
  }
}
```

After you configure LDAP to use the `memberOf` attribute, the `debug` log level will include the following log entries:

```
{"component":"authentication/v2","level":"debug","msg":"using the \"memberOf\" attribute to determine the group membership of user \"user1\"","time":"2020-06-25T14:10:58-04:00"}
{"component":"authentication/v2","level":"debug","msg":"found 1 LDAP group(s): [\"sensu\"]","time":"2020-06-25T14:10:58-04:00"}
```

LDAP specification

Top-level attributes

type

description	Top-level attribute that specifies the `sensuctl create` resource type. For LDAP definitions, the `type` should always be `ldap`.
required	true
type	String

YML

```
type: ldap
```

JSON

```
{
  "type": "ldap"
}
```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For LDAP definitions, the `api_version` should always be `authentication/v2`.
required	true
type	String

YML

```
api_version: authentication/v2
```

JSON

```
{
  "api_version": "authentication/v2"
}
```

metadata

description	Top-level map that contains the LDAP definition `name`. Review the metadata attributes for details.
required	true
type	Map of key-value pairs
example	**YML** `metadata:` `  name: openldap` **JSON** `{` `  "metadata": {` `  "name": "openldap"` `  }` `}`

spec

description	Top-level map that includes the LDAP spec attributes.
required	true
type	Map of key-value pairs
example	**YML** `spec:` `  servers:` `  - host: 127.0.0.1` `  port: 636` `  insecure: false` `  security: tls`

```
    trusted_ca_file: "/path/to/trusted-certificate-
authorities.pem"
    client_cert_file: "/path/to/ssl/cert.pem"
    client_key_file: "/path/to/ssl/key.pem"
    binding:
      user_dn: cn=binder,dc=acme,dc=org
      password: YOUR_PASSWORD
    group_search:
      base_dn: dc=acme,dc=org
      attribute: member
      name_attribute: cn
      object_class: groupOfNames
    user_search:
      base_dn: dc=acme,dc=org
      attribute: uid
      name_attribute: cn
      object_class: person
  allowed_groups: []
  groups_prefix: ldap
  username_prefix: ldap
```

JSON

```
{
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "binding": {
          "user_dn": "cn=binder,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "member",
```

```
          "name_attribute": "cn",
          "object_class": "groupOfNames"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "uid",
          "name_attribute": "cn",
          "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ldap",
    "username_prefix": "ldap"
  }
}
```

LDAP metadata attributes

name	
description	A unique string used to identify the LDAP configuration. Names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`).
required	true
type	String
example	**YML** `name: openldap` **JSON** `{ "name": "openldap" }`

LDAP spec attributes

servers	
description	The list of LDAP servers to use. During the authentication process, Sensu attempts to authenticate against each LDAP server in sequence until authentication is successful or there are no more servers to try.
required	true
type	Array
example	**YML** `servers:` `- host: 127.0.0.1` `  port: 636` `  insecure: false` `  security: tls` `  trusted_ca_file: "/path/to/trusted-certificate-authorities.pem"` `  client_cert_file: "/path/to/ssl/cert.pem"` `  client_key_file: "/path/to/ssl/key.pem"` `  binding:` `  user_dn: cn=binder,dc=acme,dc=org` `  password: YOUR_PASSWORD` `  group_search:` `  base_dn: dc=acme,dc=org` `  attribute: member` `  name_attribute: cn` `  object_class: groupOfNames` `  user_search:` `  base_dn: dc=acme,dc=org` `  attribute: uid` `  name_attribute: cn` `  object_class: person` **JSON**

```
{
  "servers": [
    {
      "host": "127.0.0.1",
      "port": 636,
      "insecure": false,
      "security": "tls",
      "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem",
      "client_cert_file": "/path/to/ssl/cert.pem",
      "client_key_file": "/path/to/ssl/key.pem",
      "binding": {
        "user_dn": "cn=binder,dc=acme,dc=org",
        "password": "YOUR_PASSWORD"
      },
      "group_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "member",
        "name_attribute": "cn",
        "object_class": "groupOfNames"
      },
      "user_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "uid",
        "name_attribute": "cn",
        "object_class": "person"
      }
    }
  ]
}
```

allowed_groups

description	An array of allowed LDAP group strings to include in the tokenized identity claim. Use to specify which groups to encode in the authentication provider's JSON Web Token (JWT) when the authenticated LDAP user is a member of many groups and the tokenized

	identity claim would be too large for correct web client operation.
required	false
type	Array of strings
example	**YML** `allowed_groups:` `- sensu-viewers` `- sensu-operators` **JSON** `{` `"allowed_groups": [` `"sensu-viewers",` `"sensu-operators"` `]` `}`

groups_prefix

description	The prefix added to all LDAP groups. Sensu appends the groups_prefix with a colon. For example, for the groups_prefix `ldap` and the group `dev`, the resulting group name in Sensu is `ldap:dev`. Use the groups_prefix when integrating LDAP groups with Sensu RBAC role bindings and cluster role bindings.
required	false
type	String
example	**YML** `groups_prefix: ldap` **JSON** `{`

```
  "groups_prefix": "ldap"
}
```

username_prefix	
description	The prefix added to all LDAP usernames. Sensu appends the username_prefix with a colon. For example, for the username_prefix `ldap` and the user `alice`, the resulting username in Sensu is `ldap:alice`. Use the username_prefix when integrating LDAP users with Sensu RBAC role bindings and cluster role bindings. Users *do not* need to provide the username_prefix when logging in to Sensu.
required	false
type	String
example	**YML** `username_prefix: ldap` **JSON** `{ "username_prefix": "ldap" }`

LDAP server attributes

host	
description	LDAP server IP address or fully qualified domain name (FQDN).
required	true
type	String

example	**YML** ```yaml host: 127.0.0.1 ``` **JSON** ```json { "host": "127.0.0.1" } ```

port

description	LDAP server port.
required	true
type	Integer
default	`389` for insecure connections; `636` for TLS connections
example	**YML** ```yaml port: 636 ``` **JSON** ```json { "port": 636 } ```

insecure

description	Skips SSL certificate verification when set to `true`. ***WARNING**: Do not use an insecure connection in production*

environments.

required	false
type	Boolean
default	`false`
example	**YML** `insecure: false` **JSON** `{ "insecure": false }`

security

description	Determines the encryption type to be used for the connection to the LDAP server: `insecure` (unencrypted connection; not recommended for production), `tls` (secure encrypted connection), or `starttls` (unencrypted connection upgrades to a secure connection).
type	String
default	`tls`
example	**YML** `security: tls` **JSON** `{ "security": "tls" }`

trusted_ca_file

description	Path to an alternative CA bundle file in PEM format to be used instead of the system's default bundle. This CA bundle is used to verify the server's certificate.
required	false
type	String
example	**YML** `trusted_ca_file: /path/to/trusted-certificate-authorities.pem` **JSON** `{ "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem" }`

client_cert_file

description	Path to the certificate that should be sent to the server if requested.
required	false
type	String
example	**YML** `client_cert_file: /path/to/ssl/cert.pem` **JSON** `{`

```
  "client_cert_file": "/path/to/ssl/cert.pem"
}
```

client_key_file

description	Path to the key file associated with the `client_cert_file`.
required	false
type	String
example	**YML** `client_key_file: /path/to/ssl/key.pem` **JSON** `{ "client_key_file": "/path/to/ssl/key.pem" }`

binding

description	The LDAP account that performs user and group lookups. If your server supports anonymous binding, you can omit the `user_dn` or `password` attributes to query the directory without credentials. Review the binding attributes for details.
required	false
type	Map
example	**YML** `binding:` `  user_dn: cn=binder,dc=acme,dc=org` `  password: YOUR_PASSWORD`

JSON

```
{
  "binding": {
    "user_dn": "cn=binder,dc=acme,dc=org",
    "password": "YOUR_PASSWORD"
  }
}
```

group_search

description	Search configuration for groups. Review the group search attributes for more information. Remove the `group_search` object from your configuration to use the `memberOf` attribute instead.
required	false
type	Map

YML

example

```
group_search:
  base_dn: dc=acme,dc=org
  attribute: member
  name_attribute: cn
  object_class: groupOfNames
```

JSON

```
{
  "group_search": {
    "base_dn": "dc=acme,dc=org",
    "attribute": "member",
    "name_attribute": "cn",
    "object_class": "groupOfNames"
  }
}
```

user_search	
description	Search configuration for users. Review the user search attributes for more information.
required	true
type	Map
example	**YML** `user_search:` `  base_dn: dc=acme,dc=org` `  attribute: uid` `  name_attribute: cn` `  object_class: person` **JSON** `{` `  "user_search": {` `  "base_dn": "dc=acme,dc=org",` `  "attribute": "uid",` `  "name_attribute": "cn",` `  "object_class": "person"` `  }` `}`

LDAP binding attributes

user_dn	
description	The LDAP account that performs user and group lookups. We recommend using a read-only account. Use the distinguished name (DN) format, such as `cn=binder,cn=users,dc=domain,dc=tld`. If your

	server supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String
example	**YML** `user_dn: cn=binder,dc=acme,dc=org` **JSON** `{ "user_dn": "cn=binder,dc=acme,dc=org" }`

password

description	Password for the `user_dn` account. If your server supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String
example	**YML** `password: YOUR_PASSWORD` **JSON** `{ "password": "YOUR_PASSWORD" }`

LDAP group search attributes

base_dn

description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String
example	**YML** `base_dn: dc=acme,dc=org` **JSON** `{ "base_dn": "dc=acme,dc=org" }`

attribute

description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false
type	String
default	`member`
example	**YML** `attribute: member` **JSON** `{`

```
  "attribute": "member"
}
```

name_attribute

description	Represents the attribute to use as the entry name.
required	false
type	String
default	`cn`
example	**YML** `name_attribute: cn` **JSON** `{ "name_attribute": "cn" }`

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`groupOfNames`
example	**YML** `object_class: groupOfNames`

JSON

```
{
  "object_class": "groupOfNames"
}
```

LDAP user search attributes

base_dn

description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String
example	**YML** `base_dn: dc=acme,dc=org` **JSON** `{ "base_dn": "dc=acme,dc=org" }`

attribute

description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false

type	String
default	`uid`
example	**YML** ```yaml attribute: uid ``` **JSON** ```json { "attribute": "uid" } ```

name_attribute

description	Represents the attribute to use as the entry name
required	false
type	String
default	`cn`
example	**YML** ```yaml name_attribute: cn ``` **JSON** ```json { "name_attribute": "cn" } ```

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`person`
example	**YML** `object_class: person` **JSON** `{ "object_class": "person" }`

LDAP troubleshooting

To troubleshoot any issue with LDAP authentication, start by increasing the log verbosity of sensu-backend to the debug log level. Most authentication and authorization errors are only displayed on the debug log level to avoid flooding the log files.

***NOTE**: If you can't locate any log entries referencing LDAP authentication, run sensuctl auth list to make sure that you successfully installed the LDAP provider.*

Authentication

This section lists common authentication error messages and describes possible solutions for each of them.

`failed to connect: LDAP Result Code 200 "Network Error"`

The LDAP provider couldn't establish a TCP connection to the LDAP server. Verify the `host` and `port` attributes. If you are not using LDAP over TLS/SSL, make sure to set the value of the `security` attribute to `insecure` for plaintext communication.

`certificate signed by unknown authority`

If you are using a self-signed certificate, make sure to set the `insecure` attribute to `true`. This will bypass verification of the certificate's signing authority.

`failed to bind: ...`

The first step for authenticating a user with the LDAP provider is to bind to the LDAP server using the service account specified in the `binding` object. Make sure the `user_dn` attribute specifies a valid **DN** and that its password is correct.

`user <username> was not found`

The user search failed. No user account could be found with the given username. Check the `user_search` object and make sure that:

- The specified `base_dn` contains the requested user entry DN
- The specified `attribute` contains the *username* as its value in the user entry
- The `object_class` attribute corresponds to the user entry object class

`ldap search for user <username> returned x results, expected only 1`

The user search returned more than one user entry, so the provider could not determine which of these entries to use. Change the `user_search` object so the provided `username` can be used to uniquely identify a user entry. Here are two methods to try:

- Adjust the `attribute` so its value (which corresponds to the `username`) is unique among the user entries
- Adjust the `base_dn` so it only includes one of the user entries

`ldap entry <DN> missing required attribute <name_attribute>`

The user entry returned (identified by `<DN>`) doesn't include the attribute specified by

`name_attribute` object, so the LDAP provider could not determine which attribute to use as the username in the user entry. Adjust the `name_attribute` so it specifies a human-readable name for the user.

`ldap group entry <DN> missing <name_attribute> and cn attributes`

The group search returned a group entry (identified by `<DN>`) that doesn't have the `name_attribute` object or a `cn` attribute, so the LDAP provider could not determine which attribute to use as the group name in the group entry. Adjust the `name_attribute` so it specifies a human-readable name for the group.

Authorization

Once authenticated, each user needs to be granted permissions via either a `ClusterRoleBinding` or a `RoleBinding`.

The way LDAP users and LDAP groups can be referred as subjects of a cluster role or role binding depends on the `groups_prefix` and `username_prefix` configuration attributes values of the LDAP provider. For example, for the groups_prefix `ldap` and the group `dev`, the resulting group name in Sensu is `ldap:dev`.

Permissions are not granted via the LDAP group(s)

During authentication, the LDAP provider will print all groups found in LDAP (for example, `found 1 group(s): [dev]`) in the logs. Keep in mind that this group name does not contain the `groups_prefix` at this point.

The Sensu backend logs each attempt made to authorize an RBAC request. This is useful for determining why a specific binding didn't grant the request. For example:

```
[...] the user is not a subject of the ClusterRoleBinding cluster-admin [...]
[...] could not authorize the request with the ClusterRoleBinding system:user [...]
[...] could not authorize the request with any ClusterRoleBindings [...]
```

OpenID Connect 1.0 protocol (OIDC) reference

***COMMERCIAL FEATURE**: Access OpenID Connect 1.0 protocol (OIDC) authentication for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool.

In addition to the built-in basic authentication, Sensu offers commercial support for single sign-on (SSO) authentication using the OpenID Connect 1.0 protocol (OIDC) on top of the OAuth 2.0 protocol. The Sensu OIDC provider is tested with Okta and PingFederate.

***WARNING**: Defining multiple OIDC providers can lead to inconsistent authentication behavior. Use `sensuctl auth list` to verify that you have defined only one authentication provider of type `OIDC`. If more than one OIDC auth provider configuration is listed, use `sensuctl auth delete $NAME` to remove the extra OIDC configurations by name.*

For general information about configuring authentication providers, read Configure single sign-on (SSO) authentication.

OIDC configuration example

YML

```
---
type: oidc
api_version: authentication/v2
metadata:
  name: oidc_name
spec:
  additional_scopes:
  - groups
  - email
```

```
  client_id: a8e43af034e7f2608780
  client_secret: b63968394be6ed2edb61c93847ee792f31bf6216
  disable_offline_access: false
  redirect_uri: http://127.0.0.1:8080/api/enterprise/authentication/v2/oidc/callback
  server: https://oidc.example.com:9031
  groups_claim: groups
  groups_prefix: 'oidc:'
  username_claim: email
  username_prefix: 'oidc:'
```

JSON

```
{
  "type": "oidc",
  "api_version": "authentication/v2",
  "metadata": {
    "name": "oidc_name"
  },
  "spec": {
    "additional_scopes": [
      "groups",
      "email"
    ],
    "client_id": "a8e43af034e7f2608780",
    "client_secret": "b63968394be6ed2edb61c93847ee792f31bf6216",
    "disable_offline_access": false,
    "redirect_uri": "http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/oidc/callback",
    "server": "https://oidc.example.com:9031",
    "groups_claim": "groups",
    "groups_prefix": "oidc:",
    "username_claim": "email",
    "username_prefix": "oidc:"
  }
}
```

OIDC specification

OIDC top-level attributes

type

description	Top-level attribute that specifies the `sensuctl create` resource type. For OIDC configuration, the `type` should always be `oidc`.
required	true
type	String

YML

```
type: oidc
```

JSON

```
{
  "type": "oidc"
}
```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For OIDC configuration, the `api_version` should always be `authentication/v2`.
required	true
type	String

YML

```
api_version: authentication/v2
```

JSON

```
{
  "api_version": "authentication/v2"
```

```
}
```

metadata

description	Top-level collection of metadata about the OIDC configuration. The `metadata` map is always at the top level of the OIDC definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope.
required	true
type	Map of key-value pairs
example	**YML** `metadata:` `  name: oidc_name` **JSON** `{` `  "metadata": {` `  "name": "oidc_name"` `  }` `}`

spec

description	Top-level map that includes the OIDC spec attributes.
required	true
type	Map of key-value pairs
example	**YML** `spec:`

```
  additional_scopes:
  - groups
  - email
  client_id: a8e43af034e7f2608780
  client_secret: b63968394be6ed2edb61c93847ee792f31bf6216
  disable_offline_access: false
  redirect_uri: http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/o
idc/callback
  server: https://oidc.example.com:9031
  groups_claim: groups
  groups_prefix: 'oidc:'
  username_claim: email
  username_prefix: 'oidc:'
```

JSON

```
{
  "spec": {
    "additional_scopes": [
      "groups",
      "email"
    ],
    "client_id": "a8e43af034e7f2608780",
    "client_secret":
"b63968394be6ed2edb61c93847ee792f31bf6216",
    "disable_offline_access": false,
    "redirect_uri": "http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/o
idc/callback",
    "server": "https://oidc.example.com:9031",
    "groups_claim": "groups",
    "groups_prefix": "oidc:",
    "username_claim": "email",
    "username_prefix": "oidc:"
  }
}
```

OIDC metadata attribute

name	
description	A unique string used to identify the OIDC configuration. The name cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). The name you choose will be used in the web UI message for OIDC sign-in: `SIGN-IN WITH <name>`.
required	true
type	String
example	**YML** `name: oidc_provider` **JSON** `{ "name": "oidc_provider" }`

OIDC spec attributes

additional_scopes	
description	Scopes to include in the claims, in addition to the default `openid` scope. ***NOTE**: For most providers, you'll want to include `groups`, `email` and `username` in this list.*
required	false

type	Array

YML

example	

```yaml
additional_scopes:
- groups
- email
- username
```

JSON

```json
{
  "additional_scopes": [
    "groups",
    "email",
    "username"
  ]
}
```

client_id

description	The OIDC provider application client ID.

> ***NOTE**: Register an application in the OIDC provider to generate a client ID. Read register an Okta application for an example.*

required	true
type	String

YML

example	

```yaml
client_id: 1c9ae3e6f3cc79c9f1786fcb22692d1f
```

JSON

```json
{
  "client_id": "1c9ae3e6f3cc79c9f1786fcb22692d1f"
```

```
}
```

client_secret

description	The OIDC provider application client secret. ***NOTE**: Register an application in the OIDC provider to generate a client ID. Read register an Okta application for an example.*
required	true
type	String
example	**YML** `client_secret: a0f2a3c1dcd5b1cac71bf0c03f2ff1bd` **JSON** `{ "client_secret": "a0f2a3c1dcd5b1cac71bf0c03f2ff1bd" }`

disable_offline_access

description	If `true`, the OIDC provider cannot include the `offline_access` scope in the authentication request. Otherwise, `false`. We recommend setting `disable_offline_access` to `false`. If set to `true`, OIDC providers cannot return a refresh token that allows users to refresh their access tokens, and users will be logged out after 5 minutes.
required	true

default	false
type	Boolean

example

YML

```
disable_offline_access: false
```

JSON

```
{
  "disable_offline_access": false
}
```

redirect_uri

description

Redirect URL to provide to the OIDC provider. Requires `/api/enterprise/authentication/v2/oidc/callback`.

> ***NOTE**: Only required for certain OIDC providers, such as Okta.*

required	false
type	String

example

YML

```
redirect_uri: http://sensu-backend.example.com:8080/api/enterprise/authentication/v2/oidc/callback
```

JSON

```
{
  "redirect_uri": "http://sensu-backend.example.com:8080/api/enterprise/authentication/v2/oidc/callback"
}
```

server

description	The location of the OIDC server you wish to authenticate against. ***NOTE**: If you configure with http, the connection will be insecure.*
required	true
type	String
example	**YML** ``` server: https://sensu.oidc.provider.example.com ``` **JSON** ``` { "server": "https://sensu.oidc.provider.example.com" } ```

groups_claim

description	The claim to use to form the associated RBAC groups. ***NOTE**: The value held by the claim must be an array of strings.*
required	false
type	String
example	**YML**

```
groups_claim: groups
```

JSON

```
{
  "groups_claim": "groups"
}
```

groups_prefix

description	The prefix added to all OIDC groups. Sensu appends the groups_prefix with a colon. For example, for the groups_prefix `okta` and the group `dev`, the resulting group name in Sensu is `okta:dev`. Use the groups_prefix when integrating OIDC groups with Sensu RBAC role bindings and cluster role bindings.
required	false
type	String
example	**YML** `groups_prefix: 'okta:'` **JSON** `{ "groups_prefix": "okta:" }`

username_claim

description	The claim to use to form the final RBAC user name.
required	true

type	String

YML

example

```
username_claim: email
```

JSON

```
{
  "username_claim": "email"
}
```

username_prefix

description	The prefix added to all OIDC usernames. Sensu appends the username_prefix with a colon. For example, for the username_prefix `okta` and the user `alice`, the resulting username in Sensu is `okta:alice`. Use the username_prefix when integrating OIDC users with Sensu RBAC role bindings and cluster role bindings. Users *do not* need to provide the username_prefix when logging in to Sensu.
required	false
type	String

YML

example

```
username_prefix: 'okta:'
```

JSON

```
{
  "username_prefix": "okta:"
}
```

Refresh tokens

No matter which OIDC provider you use, make sure to enable refresh tokens during provider configuration. If you do not enable refresh tokens in your provider configuration, Sensu will log out of the web UI, the API, and sensuctl after 5 minutes.

Register an Okta application

To use Okta for authentication, register Sensu Go as an OIDC web application. Before you start, install Sensu Go with a valid commercial license and make sure you have access to the Okta Administrator Dashboard.

Follow the steps in this section to create an Okta application and configure an Okta OIDC provider in Sensu.

Create an Okta application

***NOTE**: These instructions are based on the Okta Developer Console. The steps may be different if you are using the Okta Classic UI.*

1. In the Okta Admin Console, navigate to *Applications*: click `Applications` > `Applications`.
2. Click **Create App Integration**.
3. In the *Create a new app integration* modal window:
    - Select the sign-in method `OIDC - OpenID Connect`.
    - Select the application type `Web Application`.
4. Click **Next**.
5. In the *New Web App Integration* dialog:
    - In the *App integration name* field, enter the app name. You can also upload a logo if desired.
    - Under *Grant type*, click to select `Refresh Token` in the *Client acting on behalf of a user* list.
    - In the *Sign-in redirect URIs* field, enter `<api_url>/api/enterprise/authentication/v2/oidc/callback`. Replace `<api_url>` with your API URL, including the API port 8080.

- Under *Assignments*, click to select `Skip group assignment for now`.

6. Click **Save**.
7. Select the *Sign On* tab, scroll to the *OpenID Connect ID Token* section, and click **Edit**.
8. In the *Groups claim filter* section:
   - In the first field, enter `groups`
   - In the dropdown menu, select `Matches regex`
   - In the second field, enter `.*`
9. Click **Save**.
10. Select the *Assignments* tab and assign people and groups to your app.

Configure an Okta OIDC provider

To create your `okta` OIDC provider in Sensu:

1. For the `additional_scopes` configuration attribute, include `groups` and `email`.
2. For the `client_id` and `client_secret` values, use the *Client ID* and *Client secret*, respectively, listed under *General* > *Client Credentials* for your Okta application.
3. For the `redirect_uri` attribute, use the *Sign-in redirect URIs* value you entered in step 5 of Create an Okta application.
4. For the `server` value, use the *Okta domain* listed under *General* > *General Settings* in your Okta application.
5. Set the `disable_offline_access` attribute to your desired value (we recommend `false`).
6. Add your Okta groups to the `groups_claim` string. For example, if you have an Okta group `groups` and you set the `groups_prefix` to `okta:`, you can set up RBAC objects to mention group `okta:groups` as needed.
7. Set the `username_claim` value to `email`.
8. Specify `groups_prefix` and `username_prefix` values if desired.

Your Okta OIDC provider configuration should be similar to this example:

YML

```
---
type: oidc
api_version: authentication/v2
metadata:
  name: okta
spec:
  additional_scopes:
  - groups
  - email
  client_id: 4sd5jxiwxfvg82PoZ5d7
  client_secret: r78316494besnNCmtmEBnS47ee792f31bf6216
  redirect_uri: http://127.0.0.1:8080/api/enterprise/authentication/v2/oidc/callback
  server: https://dev-459543913.okta.com
  disable_offline_access: false
  groups_claim: groups
  username_claim: email
  groups_prefix: 'oidc:'
  username_prefix: 'oidc:'
```

JSON

```
{
  "type": "oidc",
  "api_version": "authentication/v2",
  "metadata": {
    "name": "okta"
  },
  "spec": {
    "additional_scopes": [
      "groups",
      "email"
    ],
    "client_id": "4sd5jxiwxfvg82PoZ5d7",
    "client_secret": "r78316494besnNCmtmEBnS47ee792f31bf6216",
    "redirect_uri":
"http://127.0.0.1:8080/api/enterprise/authentication/v2/oidc/callback",
    "server": "https://dev-459543913.okta.com",
    "disable_offline_access": false,
    "groups_claim": "groups",
    "username_claim": "email",
    "groups_prefix": "oidc:",
```

```
    "username_prefix": "oidc:"
  }
}
```

Configure authorization for OIDC users

Configure authorization via role-based access control (RBAC) for your OIDC users and groups by creating roles (or cluster roles) and role bindings (or cluster role bindings) that map to the user and group names.

***NOTE**: If you do not configure authorization, users will be able to log in with OIDC but will have no permissions by default.*

Use sensuctl to login with OIDC

1. Run `sensuctl login oidc`.

   ***NOTE**: You can also use `sensuctl configure` and choose the OIDC authentication method to log in to sensuctl with OIDC.*

2. If you are using a desktop, a browser will open to allow you to authenticate and log in. If a browser does not open, launch a browser to complete the login via your OIDC provider at:

   `https://<api_url>:8080/api/enterprise/authentication/v2/oidc/authorize`

OIDC troubleshooting

This section lists common OIDC errors and describes possible solutions for each of them.

To troubleshoot any issue with OIDC authentication, start by increasing the log verbosity of sensu-backend to the debug log level. Most authentication and authorization errors are only displayed on the debug log level to avoid flooding the log files.

***NOTE**: If you can't locate any log entries referencing OIDC authentication, run sensuctl auth list to make sure that you successfully installed the OIDC provider.*

For provider-specific troubleshooting, read the Okta or PingFederate documentation.

`bad request`

After configuring OIDC access, if you receive a `bad request` error when you open the web UI, you may be using an incorrect port in the redirect URI.

Make sure the redirect URI uses the API port, `8080`. Confirm that the redirect URI specified in your OIDC provider as well as in the `redirect_uri` attribute in your Sensu OIDC definition both use port `8080`. For example, the URL `http://127.0.0.1:8080/api/enterprise/authentication/v2/oidc/callback` uses the correct port.

`could not find the groups claim in the user's claims`

If you see the following error when you open the web UI, the `groups_claim` value in your Sensu OIDC definition is incorrect:

```
{"message":"could not find the groups claim \"okta:groups\" in the user's claims:
[\"sub\" \"email\" \"email_verified\"]","code":0}
```

Update your OIDC definition to specify `groups` as the value for the `groups_claim` attribute.

No namespaces or resources in the web UI after OIDC sign-in

You must configure RBAC authorization for your OIDC users and groups by creating roles (or cluster roles) and role bindings (or cluster role bindings) that map to the user and group names.

If you do not configure authorization, users will be able to log in with OIDC but will have no permissions, so they will not see any namespaces or resources in the web UI.

Inconsistent authentication

If you experience inconsistent authentication with OIDC sign-in, such as being unable to sign in after previously signing in successfully, you may have configured more than one OIDC authentication provider.

Run sensuctl auth list to make sure that you have only one authentication provider listed for type `OIDC`. If more than one OIDC authentication provider is listed, use `sensuctl auth delete $NAME` to remove the extra OIDC configuration by name.

API keys reference

API keys are long-lived authentication tokens that make it more convenient for Sensu plugins and other Sensu-adjacent applications to authenticate with the Sensu API. Unlike authentication tokens, API keys are persistent and do not need to be refreshed every 15 minutes.

The Sensu backend generates API keys, and you can provide them to applications that want to interact with the Sensu API.

Use the core/v2/apikeys API endpoints to create, retrieve, and delete API keys.

API key example

This example shows an `APIKey` resource definition:

YML

```
---
type: APIKey
api_version: core/v2
metadata:
  name: 19803eb8-36a6-4203-a225-28ec4e9f4444
spec:
  created_at: 1570732266
  username: admin
```

JSON

```
{
  "type": "APIKey",
  "api_version": "core/v2",
  "metadata" : {
    "name": "19803eb8-36a6-4203-a225-28ec4e9f4444"
  },
  "spec": {
    "created_at": 1570732266,
    "username": "admin"
```

```
    }
  }
```

Authorization header format

Use the following header format to authenticate with API keys, replacing `<API_KEY>` with your API key:

```
Authorization: Key <API_KEY>
```

This is different from the authentication token, which uses the `Authorization: Bearer` header format.

When you specify an API key in a request, the system resolves it to an authentication token and continues through the regular authentication process.

***NOTE**: The API key resource is not compatible with `sensuctl create`.*

API key specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. The `api_version` should always be `core/v2`.
required	true
type	String
example	YML `api_version: core/v2`

JSON

```
{
  "api_version": "core/v2"
}
```

metadata

description	Top-level collection of metadata about the API key, including `name` and `created_by`. The `metadata` map is always at the top level of the API key definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope.
required	true
type	Map of key-value pairs
example	**YML** `metadata:` `  name: 19803eb8-36a6-4203-a225-28ec4e9f4444` `  created_by: admin`

JSON

```
{
  "metadata": {
    "name": "19803eb8-36a6-4203-a225-28ec4e9f4444",
    "created_by": "admin"
  }
}
```

spec

description	Top-level map that includes the API key's spec attributes.

required	true
type	Map of key-value pairs
example	**YML** ```yaml spec: created_at: 1570732266 username: admin ``` **JSON** ```json { "spec": { "created_at": 1570732266, "username": "admin" } } ```

type

description	Top-level attribute that specifies the resource type. API keys should always be type `APIKey`.
required	true
type	String
example	**YML** ```yaml type: APIKey ``` **JSON** ```json { "type": "APIKey" } ```

Metadata attributes

created_by	
description	Username of the Sensu user who created the API key or last updated the API key. Sensu automatically populates the `created_by` field when the API key is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

name	
description	Unique string used to identify the API key. Sensu randomly generates a universally unique identifier (UUID) for the `name` value — users cannot provide a name for an API key.
required	true
type	String
example	**YML** `name: 19803eb8-36a6-4203-a225-28ec4e9f4444` **JSON**

```
{
  "name": "19803eb8-36a6-4203-a225-28ec4e9f4444"
}
```

Spec attributes

created_at

description	Time at which the API key was created. Unix timestamp that is automatically generated when the API key is created.
required	true
type	Integer
example	**YML** `created_at: 1234567890` **JSON** `{ "created_at": 1234567890 }`

username

description	User associated with the API key.
required	true
type	Array
example	**YML**

```
username: admin
```

JSON

```
{
  "username": "admin"
}
```

Namespaces reference

Namespaces partition resources within Sensu. Sensu entities, checks, handlers, and other namespaced resources belong to a single namespace.

Namespaces help teams use different resources (like entities, checks, and handlers) within Sensu and impose their own controls on those resources. A Sensu instance can have multiple namespaces, each with their own set of managed resources. Resource names must be unique within a namespace but do not need to be unique across namespaces.

Namespace configuration applies to sensuctl, the API, and the web UI. To create and manage namespaces, configure sensuctl as the default `admin` user or create a cluster role with `namespaces` permissions.

Namespace example

This example shows the resource definition for a `production` namespace. You can use this example with `sensuctl create` to create a `production` namespace in your Sensu deployment:

YML

```
---
type: Namespace
api_version: core/v2
metadata: {}
spec:
  name: production
```

JSON

```
{
  "type": "Namespace",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "name": "production"
  }
```

```
}
```

Best practices for namespaces

Use namespaces for isolation, not organization

Use namespaces to enforce full isolation of different groups of resources, not to organize resources. An agent cannot belong to two namespaces or execute checks in two different namespaces. To organize resources, use labels rather than multiple namespaces.

Default namespaces

Every Sensu backend includes a `default` namespace. All resources created without a specified namespace are created within the `default` namespace.

At startup, Sensu also creates the `sensu-system` namespace to contain backend entities. The `sensu-system` namespace and backend entities facilitate backend error reporting and make Sensu deployments more observable by enabling backend-generated status and metrics events.

Only cluster admins have access to the `sensu-system` namespace. If you have cluster admin permissions, you can use sensuctl and the web UI to access backend entities like other entities.

Assign a resource to a namespace

You can assign a resource to a namespace in the resource definition. Only resources that belong to a namespaced resource type (like checks, filters, and handlers) can be assigned to a namespace.

For example, to assign a check called `check-cpu` to the `production` namespace, include the `namespace` attribute in the check definition:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
```

```
  name: check-cpu
  namespace: production
spec:
  check_hooks: null
  command: check-cpu.sh -w 75 -c 90
  handlers:
  - slack
  interval: 30
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-cpu",
    "namespace": "production"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu.sh -w 75 -c 90",
    "handlers": ["slack"],
    "interval": 30,
    "subscriptions": ["system"],
    "timeout": 0,
    "ttl": 0
  }
}
```

Read the reference docs for the corresponding resource type to create resource definitions.

***PRO TIP**: If you omit the `namespace` attribute from resource definitions, you can use the `senusctl create --namespace` flag to specify the namespace for a group of resources at the time of creation. This allows you to replicate resources across namespaces without manual editing.*

Read the sensuctl documentation for more information about creating resources across namespaces and using the sensuctl namespace command.

Namespace specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. The `api_version` should always be `core/v2`.
required	true
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata	
description	Top-level collection of metadata about the namespace. For namespaces, the metatdata should always be empty.
required	true
type	Map of key-value pairs
example	**YML**

```
metadata: {}
```

JSON

```
{
  "metadata": {}
}
```

spec	
description	Top-level map that includes the namespace's spec attributes.
required	true
type	Map of key-value pairs
example	**YML** `spec:` `  name: production` **JSON** `{` `  "spec": {` `  "name": "production"` `  }` `}`

type	
description	Top-level attribute that specifies the resource type. Namespaces should always be type `Namespace`.
required	true

type	String

YML

example	`type: Namespace`

JSON

```
{
  "type": "Namespace"
}
```

Spec attributes

name

description	Name of the namespace. Namespace names can contain alphanumeric characters and hyphens and must begin and end with an alphanumeric character.
required	true
type	String

YML

example	`name: production`

JSON

```
{
  "name": "production"
}
```

Role-based access control (RBAC) reference

Sensu's role-based access control (RBAC) helps different teams and projects share a Sensu instance. Use RBAC to specify the actions users are allowed to take against specific Sensu resources, within namespaces or across all namespaces, based on roles bound to the user or to one or more groups the user is a member of.

- Roles create sets of permissions (for example, get and delete) tied to resource types. Cluster roles apply permissions across namespaces and include access to cluster-wide resources like users and namespaces.
- Users represent a person or agent that interacts with Sensu. Users can belong to one or more groups.
- Role bindings assign a role to a set of users and groups within a namespace. Cluster role bindings assign a cluster role to a set of users and groups cluster-wide.

RBAC configuration applies to sensuctl, the API, and the web UI.

Resources

Permissions within Sensu can be scoped to resource types, like checks, handlers, and users. List resource types in the rules arrays of role and cluster role definitions to configure permissions.

Namespaced resource types

Namespaced resources belong to a single namespace. You can set permissions for namespaced resources with roles and cluster roles.

Resource type	Description
`assets`	Dynamic runtime asset resources within a namespace
`checks`	Check resources within a namespace

`entities`	Entity resources within a namespace
`events`	Event resources within a namespace
`extensions`	Placeholder type
`filters`	Filter resources within a namespace
`handlers`	Handler resources within a namespace
`hooks`	Hook resources within a namespace
`mutators`	Mutator resources within a namespace
`pipelines`	Resources composed of event processing workflows
`rolebindings`	Namespace-specific role assigners
`roles`	Namespace-specific permission sets
`rule-templates`	Resources applied to service components for business service monitoring
`searches`	Saved web UI search queries
`secrets`	Secrets (for example, username or password)
`service-components`	Resources that represent elements in a business service
`silenced`	Silencing resources within a namespace
`sumo-logic-metrics-handlers`	Persistent handlers for transmitting metrics to Sumo Logic
`tcp-stream-handlers`	Persistent handlers for sending events to TCP sockets for remote storage

Cluster-wide resource types

Cluster-wide resources cannot be assigned to a namespace. You can set permissions for cluster-wide resources only with cluster roles.

Resource type	Description
`apikeys`	Persistent universally unique identifier (UUID) for authentication
`authproviders`	Authentication provider configuration
`clusterrolebindings`	Cluster-wide role assigners
`clusterroles`	Cluster-wide permission sets
`clusters`	Sensu clusters running multiple Sensu backends
`config`	Global configuration for web UI display
`etcd-replicators`	Mirror RBAC resource changes to follower clusters
`license`	Sensu commercial license
`namespaces`	Resource partitions within a Sensu instance
`provider`	PostgreSQL event store provider
`providers`	Secrets providers
`users`	People or agents that interact with Sensu

Special resource types

You can set permissions for special resource types with roles and cluster roles.

Type	Description
`*`	All resources within Sensu. **The `*` type takes precedence over other rules within the same role.** If you want to deny a certain type, you can't use the `*` type. Instead, you must explicitly allow every type required. When applied to a role, the `*` type applies only to namespaced resource types. When applied to a cluster role, the `*` type applies to both namespaced resource types and cluster-wide resource types.

Users

A user represents a person or an agent that interacts with Sensu.

You can assign users to one or more roles or cluster roles. You can also assign users to one or more groups. Users inherit all permissions from each role or cluster role they are assigned to, whether they are assigned as users or as a member of a group.

Users can use their assigned Sensu username and password to configure sensuctl and log in to the web UI.

User example

The following example shows a user resource definition:

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  groups:
  - ops
  - dev
  password: user_password
  password_hash: $5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm
  username: alice
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "username": "alice",
    "password": "user_password",
    "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm",
```

```
    "disabled": false,
    "groups": ["ops", "dev"]
  }
}
```

To create this user with `sensuctl create`, first, save the definition to a file like `users.yml` or `users.json`. Then, run:

SHELL

```
sensuctl create --file users.yml
```

SHELL

```
sensuctl create --file users.json
```

Default users

Sensu automatically creates an administrator user and an `agent` user during installation.

Administrator user

During the Sensu backend installation process, you create a username and password for an `admin` user.

The `admin` user is automatically added to the `cluster-admins` group and the `cluster-admin` cluster role, which are both listed in the cluster role binding `cluster-admin`. The group, cluster role, and cluster role binding assignments give the `admin` user permissions to manage all aspects of Sensu, as well as create new users.

After you configure sensuctl, you can change the `admin` user's password with the `change-password` command.

`agent` *user*

Sensu creates a default `agent` user with the password `P@ssw0rd!` during startup. The user/password combination corresponds to the defaults the Sensu agent uses.

By default, the `agent` user belongs to the `system:agent` group. The `system:agent` cluster role binding grants the `system:agent` cluster role to the members of this group. To grant agent users the permissions they need to report events into any namespace, add agent users to the `system:agent` group.

Configure the `agent` user's credentials with the `user` and `password` agent configuration options.

View users

Use sensuctl to list all users within Sensu.

To return a list of users in `yaml` or `wrapped-json` format for use with `sensuctl create`:

SHELL

```
sensuctl user list --format yaml
```

SHELL

```
sensuctl user list --format wrapped-json
```

Test and change user passwords

To test the password for a user created with Sensu's built-in basic authentication, run:

```
sensuctl user test-creds <USERNAME> --password '<PASSWORD>'
```

An empty response indicates the user's password is valid. A `request-unauthorized` response indicates the user's password is invalid.

***NOTE**: The `sensuctl user test-creds` command tests passwords for users created with Sensu's built-in basic authentication. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).*

To change a user's password:

```
sensuctl user change-password <USERNAME> --current-password <CURRENT_PASSWORD> --
new-password <NEW_PASSWORD>
```

You can also use sensuctl to reset a user's password or generate a password hash.

Create users

You can use sensuctl to create users. For example, the following command creates a user with the username `alice`, creates a password, and assigns the user to the `ops` and `dev` groups:

```
sensuctl user create alice --password='<PASSWORD>' --groups=ops,dev
```

> ***NOTE**: Passwords must have at least eight characters.*

You can create any number of users, each with their own passwords. As a general rule, users have no permissions by default. Users are granted permissions by role bindings or cluster role bindings.

Disable users

To disable a user, run:

```
sensuctl user disable <USERNAME>
```

To reinstate a disabled user, run:

```
sensuctl user reinstate <USERNAME>
```

Assign user permissions

To assign permissions to a user:

1. Create the user.
2. Create a role (or a cluster role for cluster-wide access).
3. Create a role binding (or cluster role binding) to assign the role to the user.

Groups

A group is a set of users within Sensu. You can assign groups to one or more roles, and users can belong to one or more groups.

Groups inherit all permissions from each role they are assigned to.

> ***NOTE**: Groups are not a resource type within Sensu. Instead, groups are created and managed only within user definitions.*

Default groups

Sensu includes a default `cluster-admins` group that contains the default `admin` user and a `system:agents` group used internally by Sensu agents.

Add groups to users

Use sensuctl to add a group to a user:

```
sensuctl user add-group <USERNAME> <GROUP>
```

You can also set a user's list of groups to a specific list:

```
sensuctl user set-groups <USERNAME> <GROUP1>[,<GROUP2>, ...<GROUP2>]
```

Remove groups from users

Use sensuctl to remove groups from users.

To remove a group from a user:

```
sensuctl user remove-group <USERNAME> <GROUP>
```

To remove all groups from a user:

```
sensuctl user remove-groups <USERNAME>
```

Roles

A role is a set of permissions that control access to Sensu resources within a single namespace. Use role bindings to assign roles to users and groups.

To create and manage roles within a single namespace, create a role with `roles` permissions within that namespace. To create and manage roles cluster-wide, configure sensuctl as the default `admin` user or create a cluster role with `roles` permissions.

To avoid recreating commonly used roles in every namespace, create a cluster role and use a role binding (not a cluster role binding) to restrict permissions within a specific namespace.

Role example

The following example shows a role resource definition:

YML

```
---
type: Role
api_version: core/v2
metadata:
  name: namespaced-resources-all-verbs
spec:
  rules:
```

```
  - resources:
    - assets
    - checks
    - entities
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - pipelines
    - rolebindings
    - roles
    - silenced
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "namespaced-resources-all-verbs"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
```

```
            "pipelines",
            "rolebindings",
            "roles",
            "silenced",
            "sumo-logic-metrics-handlers",
            "tcp-stream-handlers"
          ],
          "verbs": [
            "get",
            "list",
            "create",
            "update",
            "delete"
          ]
        }
      ]
    }
  }
```

To create this role with `sensuctl create`, first save the definition to a file like `roles.yml` or `roles.json`.

Then, run:

SHELL

```
sensuctl create --file roles.yml
```

SHELL

```
sensuctl create --file roles.json
```

Default roles

Every Sensu backend includes the `system:pipeline` role, which is a facility that allows the EventFilter engine to load events from Sensu's event store. The `system:pipeline` role is an implementation detail and should not be assigned to Sensu users.

View roles

Use sensuctl to list all roles within Sensu:

```
sensuctl role list
```

To review the permissions and scope for a specific role:

```
sensuctl role info admin
```

To get help managing roles with sensuctl:

```
sensuctl role help
```

Edit roles

To edit a role:

```
sensuctl edit role <ROLE> <flags>
```

To get more information about available flags, run:

```
sensuctl edit --help
```

Create roles

You can use sensuctl to create roles.Read Create a role and role binding for an example.

Delete roles

To delete a role:

```
sensuctl role delete <ROLE>
```

Cluster roles

A cluster role is a set of permissions that control access to Sensu resources. Cluster roles can include permissions for cluster-wide resources in addition to namespaced resources.

You can also use cluster roles (in conjunction with cluster role bindings) to grant access to namespaced resources across all namespaces. This allows you to run commmands like `sensuctl check list --all-namespaces`.

To create and manage cluster roles, configure sensuctl as the default `admin` user or create a cluster role with permissions for `clusterroles`. To create and manage roles cluster-wide, configure sensuctl as the default `admin` user or create a cluster role with `roles` permissions.

To avoid recreating commonly used roles in every namespace, create a cluster role and use a role binding (not a cluster role binding) to restrict permissions within a specific namespace.

Cluster role example

The following example shows a cluster role resource definition:

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: all-resources-all-verbs
spec:
  rules:
  - resources:
    - assets
    - checks
    - entities
```

```
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - pipelines
    - rolebindings
    - roles
    - silenced
    - cluster
    - clusterrolebindings
    - clusterroles
    - namespaces
    - users
    - authproviders
    - license
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "all-resources-all-verbs"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
```

```
            "handlers",
            "hooks",
            "mutators",
            "pipelines",
            "rolebindings",
            "roles",
            "silenced",
            "cluster",
            "clusterrolebindings",
            "clusterroles",
            "namespaces",
            "users",
            "authproviders",
            "license",
            "sumo-logic-metrics-handlers",
            "tcp-stream-handlers"
          ],
          "verbs": [
            "get",
            "list",
            "create",
            "update",
            "delete"
          ]
        }
      ]
    }
}
```

To create this cluster role with `sensuctl create`, first save the definition to a file like `cluster_roles.yml` or `cluster_roles.json`. Then, run:

SHELL

```
sensuctl create --file cluster_roles.yml
```

SHELL

```
sensuctl create --file cluster_roles.json
```

Default cluster roles

Every Sensu backend includes the following cluster roles:

Cluster role name	Description
`cluster-admin`	Full access to all resource types across namespaces, including access to cluster-wide resource types.
`admin`	Full access to all resource types. Apply this cluster role within a namespace with a role binding (not a cluster role binding).
`edit`	Read and write access to most resource types except roles and role bindings. Apply this cluster role within a namespace with a role binding (not a cluster role binding).
`view`	Read-only access to most resource types except roles and role bindings. Apply this cluster role within a namespace with a role binding (not a cluster role binding).
`system:agent`	Used internally by Sensu agents. Configure an agent's user credentials with the `user` and `password` agent configuration flags.
`system:user`	Get and update permissions for local resources for the current user.

View cluster roles

Use sensuctl to list all cluster roles within Sensu:

```
sensuctl cluster-role list
```

To review the permissions and scope for a specific cluster role:

```
sensuctl cluster-role info <CLUSTER-ROLE>
```

To get help managing roles with sensuctl:

```
sensuctl cluster-role help
```

Create cluster roles

You can use sensuctl to create cluster roles.Read Create a cluster role and cluster role binding for an example.

Delete cluster roles

To delete a cluster role:

```
sensuctl cluster-role delete <CLUSTER-ROLE>
```

Role bindings

A role binding assigns a role or a cluster role to users and groups within a single namespace.

To create and manage role bindings within a namespace, create a role with `rolebindings` permissions within that namespace, and log in by configuring sensuctl.

Without an assigned role or cluster role, users can sign in to the web UI but can't access any Sensu resources.With the correct roles and bindings configured, users can log in to sensuctl and the web UI using their single-sign-on username and password (no prefixes required).

Make sure to include the groups_prefix and username_prefix for the authentication provider when you create Sensu role bindings.

Role binding example

The following example shows a role binding resource definition:

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: event-reader-binding
spec:
  role_ref:
    name: event-reader
    type: Role
  subjects:
  - name: bob
    type: User
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "event-reader-binding"
  },
  "spec": {
    "role_ref": {
      "name": "event-reader",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "bob",
        "type": "User"
      }
    ]
  }
}
```

To create this role binding with `sensuctl create`, first save the definition to a file like `rolebindings.yml` or `rolebindings.json`. Then, run:

SHELL

```
sensuctl create --file rolebindings.yml
```

SHELL

```
sensuctl create --file rolebindings.json
```

Default role bindings

Every Sensu backend includes the `system:pipeline` role binding, a facility that allows the EventFilter engine to load events from Sensu's event store. The `system:pipeline` role binding is an implementation detail and should not be applied to Sensu users. |

View role bindings

Use sensuctl to list all role bindings within Sensu:

```
sensuctl role-binding list
```

To review the details for a specific role binding:

```
sensuctl role-binding info <ROLE-BINDING>
```

To get help managing role bindings with sensuctl:

```
sensuctl role-binding help
```

Create role bindings

You can use sensuctl to create role bindings that assign a role to users and groups. Read Create a role and role binding for an example.

Delete role bindings

To delete a role binding:

```
sensuctl role-binding delete <ROLE-BINDING>
```

Cluster role bindings

A cluster role binding assigns a cluster role to users and groups across namespaces and resource types.

To create and manage cluster role bindings, configure sensuctl as the default `admin` user or create a cluster role with permissions for `clusterrolebindings`.

Without an assigned role or cluster role, users can sign in to the web UI but can't access any Sensu resources. With the correct roles and bindings configured, users can log in to sensuctl and the web UI using their single-sign-on username and password (no prefixes required).

Make sure to include the groups_prefix and username_prefix for the authentication provider when creating Sensu cluster role bindings.

Cluster role binding example

The following example shows a cluster role binding resource definition:

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: cluster-admin
spec:
  role_ref:
    name: cluster-admin
    type: ClusterRole
  subjects:
  - name: cluster-admins
```

```
    type: Group
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "cluster-admin"
  },
  "spec": {
    "role_ref": {
      "name": "cluster-admin",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "cluster-admins",
        "type": "Group"
      }
    ]
  }
}
```

To create this cluster role binding with `sensuctl create`, first save the definition to a file like `clusterrolebindings.yml` or `clusterrolebindings.json`. Then, run:

SHELL

```
sensuctl create --file clusterrolebindings.yml
```

SHELL

```
sensuctl create --file clusterrolebindings.json
```

Default cluster role bindings

Every Sensu backend includes the following cluster role bindings:

Cluster role binding name	Description
`cluster-admin`	`ClusterRoleBinding`
`system:agent`	`ClusterRoleBinding`
`system:user`	`ClusterRoleBinding`

View cluster role bindings

Use sensuctl to list all cluster role bindings within Sensu:

```
sensuctl cluster-role-binding list
```

To review the details for a specific role binding:

```
sensuctl cluster-role-binding info <CLUSTER-ROLE-BINDING>
```

To get help managing cluster role bindings with sensuctl:

```
sensuctl cluster-role-binding help
```

Create cluster role bindings

You can use sensuctl to create cluster role bindings that assign cluster roles to users and groups.Read Create a cluster role and cluster role binding for an example.

Delete cluster role bindings

To delete a role binding:

```
sensuctl cluster-role-binding delete <CLUSTER-ROLE-BINDING>
```

Create a role and role binding

This example will create a role and a role binding that assigns the role to a group. As a result, all users who are assigned the group will have get, list, create, update, and delete permissions for all resources in the production namespace.

The following command creates a `prod-admin` role restricted to the production namespace:

```
sensuctl role create prod-admin --verb='get,list,create,update,delete' --
resource='*' --namespace production
```

The command creates the following role resource definition:

YML

```
---
type: Role
api_version: core/v2
metadata:
  name: prod-admin
  namespace: production
spec:
  rules:
  - resources:
    - '*'
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
```

```
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "prod-admin",
    "namespace": "production"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "*"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

Run the following command to create a role binding (or cluster role binding) to assign the `prod-admin` role created above to a group named `oncall`:

```
sensuctl role-binding create prod-admin-oncall --role=prod-admin --group=oncall
```

This command creates the following role binding resource definition:

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: prod-admin-oncall
spec:
```

```
  role_ref:
    name: prod-admin
    type: Role
  subjects:
  - name: oncall
    type: Group
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "prod-admin-oncall"
  },
  "spec": {
    "role_ref": {
      "name": "prod-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "oncall",
        "type": "Group"
      }
    ]
  }
}
```

Role bindings can also assign cluster roles to users and groups within a single namespace. For example, to create a role binding that assigns the `global-event-reader` cluster role to the user `angela` and the `event-readers` group, run:

```
sensuctl role-binding create event-readers-binding --cluster-role=global-event-reader --user=angela --group=read-events-only
```

This command creates a role binding resource definition similar to the following:

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: event-readers-binding
  namespace: default
spec:
  role_ref:
    name: global-event-reader
    type: ClusterRole
  subjects:
  - name: read-events-only
    type: Group
  - name: angela
    type: User
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "event-readers-binding",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "global-event-reader",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "read-events-only",
        "type": "Group"
      },
      {
        "name": "angela",
        "type": "User"
      }
    ]
```

```
    }
  }
```

Create a role and role binding with a group prefix

In this example, if a groups_prefix of `ad` is configured for Active Directory authentication, the role and role binding will give a `dev` group access to create and manage Sensu workflows within the `default` namespace:

YML

```
---
type: Role
api_version: core/v2
metadata:
  name: workflow-creator
spec:
  rules:
  - resources:
    - checks
    - hooks
    - filters
    - events
    - filters
    - mutators
    - pipelines
    - handlers
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
```

```
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "workflow-creator"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "checks",
          "hooks",
          "filters",
          "events",
          "filters",
          "mutators",
          "pipelines",
          "handlers",
          "sumo-logic-metrics-handlers",
          "tcp-stream-handlers"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: dev-binding-with-groups-prefix
spec:
  role_ref:
    name: workflow-creator
```

```
    type: Role
  subjects:
  - name: ad:dev
    type: Group
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "dev-binding-with-groups-prefix"
  },
  "spec": {
    "role_ref": {
      "name": "workflow-creator",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "ad:dev",
        "type": "Group"
      }
    ]
  }
}
```

Create a cluster role and cluster role binding

This example will create a cluster role and a cluster role role binding that assigns the cluster role to a user and a group. As a result, the individual user and all users who are assigned the group will have read-only access to events (and only events) across all namespaces in Sensu.

For example, the following command creates a `global-event-reader` cluster role that can read events in all namespaces:

```
sensuctl cluster-role create global-event-reader --verb='get,list' --
resource='events'
```

The command creates the following cluster role resource definition:

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: global-event-reader
spec:
  rules:
  - resources:
    - events
    verbs:
    - get
    - list
```

JSON

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "global-event-reader"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "events"
        ],
        "verbs": [
          "get",
          "list"
        ]
      }
    ]
  }
}
```

Next, run the following command to assign the `global-event-reader` cluster role to the user `angela` and the group `global-event-readers`:

```
sensuctl cluster-role-binding create global-event-reader-binding --cluster-
role=global-event-reader --user=angela --group=global-event-readers
```

This command creates a cluster role binding resource definition similar to the following:

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: global-event-reader-binding
spec:
  role_ref:
    name: global-event-reader
    type: ClusterRole
  subjects:
  - name: global-event-readers
    type: Group
  - name: angela
    type: User
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "global-event-reader-binding"
  },
  "spec": {
    "role_ref": {
      "name": "global-event-reader",
      "type": "ClusterRole"
    },
```

```
    "subjects": [
      {
        "name": "global-event-readers",
        "type": "Group"
      },
      {
        "name": "angela",
        "type": "User"
      }
    ]
  }
}
```

Assign user permissions within a namespace

To assign permissions to a user:

1. Create the user.
2. Create a role.
3. Create a role binding to assign the role to the user.

For example, the following configuration creates a user `alice`, a role `default-admin`, and a role binding `alice-default-admin`, giving `alice` full permissions for namespaced resource types within the `default` namespace. You can add these resources to Sensu using `sensuctl create`.

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
  password: user_password
```

JSON

```
{
```

```
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice",
    "password": "user_password"
  }
}
```

YML

```
---
type: Role
api_version: core/v2
metadata:
  name: default-admin
spec:
  rules:
  - resources:
    - assets
    - checks
    - entities
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - pipelines
    - rolebindings
    - roles
    - searches
    - silenced
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "pipelines",
          "rolebindings",
          "roles",
          "searches",
          "silenced",
          "sumo-logic-metrics-handlers",
          "tcp-stream-handlers"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: alice-default-admin
spec:
  role_ref:
    name: default-admin
    type: Role
  subjects:
  - name: alice
    type: User
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "alice-default-admin"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "alice",
        "type": "User"
      }
    ]
  }
}
```

Assign group permissions within a namespace

To assign permissions to group of users:

1. Create at least one user assigned to a group.
2. Create a role.
3. Create a role binding to assign the role to the group.

For example, the following configuration creates a user `alice` assigned to the group `ops`, a role `default-admin`, and a role binding `ops-default-admin`, giving the `ops` group full permissions for namespaced resource types within the `default` namespace. You can add these resources to Sensu using `sensuctl create`.

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
  password: user_password
  groups:
  - ops
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice",
    "password": "user_password",
    "groups": [
      "ops"
    ]
  }
}
```

YML

```
---
type: Role
api_version: core/v2
metadata:
  name: default-admin
spec:
  rules:
  - resources:
    - assets
    - checks
    - entities
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - pipelines
    - rolebindings
    - roles
    - searches
    - silenced
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin"
  },
  "spec": {
    "rules": [
      {
        "resources": [
```

```
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "pipelines",
          "rolebindings",
          "roles",
          "searches",
          "silenced",
          "sumo-logic-metrics-handlers",
          "tcp-stream-handlers"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: ops-default-admin
spec:
  role_ref:
    name: default-admin
    type: Role
  subjects:
  - name: ops
```

```
    type: Group
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-default-admin"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

***PRO TIP**: To avoid recreating commonly used roles in each namespace, create a cluster role and use a role binding to restrict permissions within a specific namespace.*

Assign group permissions across all namespaces

To assign cluster-wide permissions to group of users:

1. Create at least one user assigned to a group.
2. Create a cluster role.
3. Create a cluster role binding to assign the role to the group.

For example, the following configuration creates a user `alice` assigned to the group `ops`, a cluster role `default-admin`, and a cluster role binding `ops-default-admin`, giving the `ops` group full

permissions for namespaced resource types and cluster-wide resource types across all namespaces. You can add these resources to Sensu using `sensuctl create`.

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
  password: user_password
  groups:
  - ops
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice",
    "password": "user_password",
    "groups": [
      "ops"
    ]
  }
}
```

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: default-admin
spec:
  rules:
```

```
  - resources:
    - assets
    - checks
    - entities
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - pipelines
    - rolebindings
    - roles
    - silenced
    - cluster
    - clusterrolebindings
    - clusterroles
    - namespaces
    - users
    - authproviders
    - license
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin"
  },
  "spec": {
    "rules": [
      {
        "resources": [
          "assets",
```

```
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "pipelines",
          "rolebindings",
          "roles",
          "silenced",
          "cluster",
          "clusterrolebindings",
          "clusterroles",
          "namespaces",
          "users",
          "authproviders",
          "license",
          "sumo-logic-metrics-handlers",
          "tpc-stream-handlers"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: ops-default-admin
spec:
```

```
  role_ref:
    name: default-admin
    type: ClusterRole
  subjects:
  - name: ops
    type: Group
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-default-admin"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

Assign different permissions for different resource types

You can assign different permissions for different resource types in a role or cluster role definition. To do this, you'll still create at least one user assigned to a group, a role or cluster role, and a role binding or cluster role binding. However, in this case, the role or cluster role will include more than one rule.

For example, you may want users in a testing group to be able to get and list all resource types but create, update, and delete only silenced entries across all namespaces. Create a user `alice` assigned to the group `ops_testing`, a cluster role `manage_silences` with two rules (one for all resources and one just for silences), and a cluster role binding `ops_testing_manage_silences`:

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
  password: user_password
  groups:
  - ops_testing
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice",
    "password": "user_password",
    "groups": [
      "ops_testing"
    ]
  }
}
```

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: manage_silences
spec:
  rules:
  - verbs:
    - get
    - list
```

```
    resources:
    - '*'
  - verbs:
    - create
    - update
    - delete
    resources:
    - silenced
```

JSON

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "manage_silences"
  },
  "spec": {
    "rules": [
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "*"
        ]
      },
      {
        "verbs": [
          "create",
          "update",
          "delete"
        ],
        "resources": [
          "silenced"
        ]
      }
    ]
  }
}
```

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: ops_testing_manage_silences
spec:
  role_ref:
    name: manage_silences
    type: ClusterRole
  subjects:
  - name: ops_testing
    type: Group
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops_testing_manage_silences"
  },
  "spec": {
    "role_ref": {
      "name": "manage_silences",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "ops_testing",
        "type": "Group"
      }
    ]
  }
}
```

Create as many rules as you need in the role or cluster role. For example, you can configure a role or cluster role that includes one rule for each verb, with each rule listing only the resources that verb

should apply to.

Here's another example that includes three rules. Each rule specifies different access permissions for the resource types listed in the rule. In addition, the user group would have no access at all for the two resources that are not listed: API keys and licences.

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
  password: user_password
  groups:
  - ops
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice",
    "password": "user_password",
    "groups": [
      "ops"
    ]
  }
}
```

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
```

```
  name: ops_access
spec:
  rules:
  - verbs:
    - get
    - list
    resources:
    - entities
    - events
    - rolebindings
    - roles
    - clusterrolebindings
    - clusterroles
    - config
    - users
  - verbs:
    - get
    - list
    - create
    - update
    - delete
    resources:
    - assets
    - checks
    - filters
    - handlers
    - hooks
    - mutators
    - pipelines
    - rule-templates
    - searches
    - secrets
    - service-components
    - silenced
    - sumo-logic-metrics-handlers
    - tcp-stream-handlers
    - clusters
    - etcd-replicators
    - providers
  - verbs:
    - get
    - list
```

```
    - create
    - update
    resources:
    - authproviders
    - namespaces
    - provider
```

JSON

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops_access"
  },
  "spec": {
    "rules": [
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "entities",
          "events",
          "rolebindings",
          "roles",
          "clusterrolebindings",
          "clusterroles",
          "config",
          "users"
        ]
      },
      {
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ],
        "resources": [
```

```
          "assets",
          "checks",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "pipelines",
          "rule-templates",
          "searches",
          "secrets",
          "service-components",
          "silenced",
          "sumo-logic-metrics-handlers",
          "tcp-stream-handlers",
          "clusters",
          "etcd-replicators",
          "providers"
        ]
      },
      {
        "verbs": [
          "get",
          "list",
          "create",
          "update"
        ],
        "resources": [
          "authproviders",
          "namespaces",
          "provider"
        ]
      }
    ]
  }
}
```

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
```

```
metadata:
  name: ops_access_assignment
spec:
  role_ref:
    name: ops_access
    type: ClusterRole
  subjects:
  - name: ops
    type: Group
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops_access_assignment"
  },
  "spec": {
    "role_ref": {
      "name": "ops_access",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

Reuse cluster roles across namespaces

Reusing cluster roles across namespaces can reduce the number of resources you need to manage.

For example, suppose you have three teams, each with its own namespace. You write a script that uses limited service accounts to create and delete silences. You want to use the script for all three team namespaces, so you create a role with the required permissions and a role binding in each

namespace: six new resources. If you need to change the permissions for the script, you will need to update each role in the team namespaces (three resources).

A better approach is to create a single cluster role that grants the required permissions, plus one role binding in each namespace to tie the permissions to the namespace's limited service account. With this configuration, you only need to update one resource to make permission changes: the `silencing-script` cluster role. Sensu will automatically apply updates in each team's namespace using the role bindings that define each limited service account as a subject of the cluster role.

1. Create a limited service account user in each namespace:

```
sensuctl user create silencing-service-team-1 --password='password'
```

This creates the following user definition:

YML

```
---
type: User
api_version: core/v2
metadata:
  name: silencing-service-team-1
spec:
  disabled: false
  username: silencing-service-team-1
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {
    "name": "silencing-service-team-1"
  },
  "spec": {
    "disabled": false,
    "username": "silencing-service-team-1"
  }
}
```

Repeat this step to create a limited service account user in each team's namespace.

2. Create a cluster role with get, list, create, update, and delete permissions for silences:

```
sensuctl cluster-role create silencing-script --verb
get,list,create,update,delete --resource silenced
```

This command creates the cluster role that has the permissions the silencing service accounts will need:

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: silencing-script
spec:
  rules:
  - resources:
    - silenced
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "silencing-script"
  },
  "spec": {
    "rules": [
      {
```

```
        "resources": [
          "silenced"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

3. Create a role binding in each team namespace to assign the `silencing-script` cluster role to the team's `silencing-service` user. For example, use this command to create the role binding for Team 1:

```
sensuctl role-binding create silencing-script-binding-team-1 --cluster-role
silencing-script --user silencing-service-team-1 --namespace team1
```

This command creates the role binding that ties the correct permissions (via the `silencing-script` cluster role) with your service account (via the user `silencing-service-team-1`):

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  name: silencing-script-binding-team-1
spec:
  role_ref:
    name: silencing-script
    type: ClusterRole
  subjects:
  - name: silencing-service-team-1
    type: User
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "silencing-script-binding-team-1"
  },
  "spec": {
    "role_ref": {
      "name": "silencing-script",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "silencing-service-team-1",
        "type": "User"
      }
    ]
  }
}
```

Repeat this step to create a role binding for the `silencing-script` cluster role and the limited service account user in each team's namespace.

User specification

Top-level attributes for user resources

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For users in this version of Sensu, this attribute should always be `core/v2`.
required	Required for user definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.

type	String

YML

example	`api_version: core/v2`

JSON

```
{
  "api_version": "core/v2"
}
```

metadata

description	Top-level collection of metadata about the user, including `name`. The `metadata` map is always at the top level of the user definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for user resources for details.
required	Required for user definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

```
metadata:
  name: alice
```

JSON

```
{
  "metadata": {
    "name": "alice"
  }
}
```

spec	
description	Top-level map that includes the user spec attributes.
required	Required for user definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `spec:` `  disabled: false` `  groups:` `  - ops` `  - dev` `  password: user_password` `  password_hash: $5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm` `  username: alice` **JSON** `{` `  "spec": {` `  "disabled": false,` `  "groups": [` `  "ops",` `  "dev"` `  ],` `  "password": "user_password",` `  "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm",` `  "username": "alice"` `  }` `}`

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Users should always be type `User`.
required	Required for user definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

```
type: User
```

JSON

```
{
  "type": "User"
}
```

Metadata attributes for user resources

name	
description	Unique string used to identify the user. User resource names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each user resource must have a unique name.
required	true
type	String

YML

```
name: alice
```

JSON

```
{
  "name": "alice"
}
```

Spec attributes for user resources

disabled

description	If `true`, the user's account is disabled. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	**YML** `disabled: false` **JSON** `{ "disabled": false }`

groups

description	Groups to which the user belongs.
required	false
type	Array
example	**YML**

```
groups:
- dev
- ops
```

JSON

```
{
  "groups": [
    "dev",
    "ops"
  ]
}
```

password

description	User's password. Passwords must have at least eight characters. ***NOTE**: You only need to set either the `password` or the `password_hash` (not both). We recommend using the `password_hash` because it eliminates the need to store cleartext passwords.*
required	true
type	String
example	**YML** `password: user_password` **JSON** `{ "password": "user_password" }`

password_hash

description

Bcrypt password hash. You can use the `password_hash` in your user definitions instead of storing cleartext passwords.

> ***NOTE**: You only need to set either the `password` or the `password_hash` (not both). We recommend using the `password_hash` because it eliminates the need to store cleartext passwords.*

required

false

type

String

example

YML

```
password_hash:
$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyN
m
```

JSON

```
{
  "password_hash":
"$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAy
Nm"
}
```

username

description

Name of the user. Cannot contain special characters.

required

true

type

String

example	**YML** `username: alice` **JSON** `{ "username": "alice" }`

Role and cluster role specification

Top-level attributes for role and cluster role resources

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For role and cluster role resources in this version of Sensu, this attribute should always be `core/v2`.
required	Required for role and cluster role definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata	
description	Top-level collection of metadata about the role or cluster role. The `metadata` map is always at the top level of the role or cluster role definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for role and cluster role resources for details. ***NOTE**: Cluster role definitions do not include a `namespace` attribute in the resource metadata.*
required	Required for role definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  annotations:` `  managed-by: prod-admin` `  created_by: admin` `  labels:` `  environment: prod1` `  region: us-west-1` `  sensu.io/managed_by: sensuctl` `  name: prod-user` `  namespace: production` **JSON** `{` `  "metadata": {` `  "annotations": {` `  "managed-by": "prod-admin"` `  },` `  "created_by": "admin",` `  "labels": {` `  "environment": "prod1",`

```
      "region": "us-west-1",
      "sensu.io/managed_by": "sensuctl"
    },
    "name": "prod-user",
    "namespace": "production"
  }
}
```

spec	
description	Top-level map that includes the role or cluster role spec attributes.
required	Required for role or cluster role definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	YML ```yaml spec: rules: - resource_names: null resources: - checks - entities - events - filters - handlers - hooks - mutators - pipelines - searches - service-components - silenced - sumo-logic-metrics-handlers - tcp-stream-handlers verbs: - get - list - create ```

```
    - update
    - delete
```

JSON

```
{
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "pipelines",
          "searches",
          "service-components",
          "silenced",
          "sumo-logic-metrics-handlers",
          "tcp-stream-handlers"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

type

description	Top-level attribute that specifies the `sensuctl create` resource type. Roles should always be type `Role`. Cluster roles should always be type `ClusterRole`.
required	Required for role and cluster role definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example (role)	**YML** `type: Role` **JSON** `{ "type": "Role" }`
example (cluster role)	**YML** `type: ClusterRole` **JSON** `{ "type": "ClusterRole" }`

Metadata attributes for role and cluster role resources

annotations

description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu.

	In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: prod-admin` **JSON** `{` `  "annotations": {` `  "managed-by": "prod-admin"` `  }` `}`

created_by

description	Username of the Sensu user who created or last updated the role or cluster role. Sensu automatically populates the `created_by` field when the role or cluster role is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{`

```
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example

YML

```
labels:
  environment: prod1
  region: us-west-1
  sensu.io/managed_by: sensuctl
```

JSON

```
{
  "labels": {
    "environment": "prod1",
    "region": "us-west-1",
    "sensu.io/managed_by": "sensuctl"
  }
}
```

name

description	Unique string used to identify the role or cluster role. Role and cluster role names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each role must have a unique name within its namespace. Each cluster role must have a unique name.
required	true
type	String

YML

example

```
name: prod-user
```

JSON

```
{
  "name": "prod-user"
}
```

namespace

description	Sensu RBAC namespace that the role belongs to. ***NOTE**: Cluster role definitions do not include a `namespace` attribute in the resource metadata.*
required	false
type	String
default	`default`

YML

example

```
namespace: production
```

JSON

```
{
  "namespace": "production"
}
```

Spec attributes for role and cluster role resources

rules	
description	Rule set that the role or cluster role applies. A rule is an explicit statement that grants a particular access to a resource. Read rules attributes for more information.
required	true
type	Array

YML

example

```
rules:
- resource_names:
  - check-cpu
  resources:
  - checks
  - entities
  - events
  - filters
  - handlers
  - hooks
  - mutators
  - pipelines
  - searches
  - service-components
  - silenced
```

```
  - sumo-logic-metrics-handlers
  - tcp-stream-handlers
  verbs:
  - get
  - list
  - create
  - update
  - delete
```

JSON

```
{
  "rules": [
    {
      "resource_names": [
        "check-cpu"
      ],
      "resources": [
        "checks",
        "entities",
        "events",
        "filters",
        "handlers",
        "hooks",
        "mutators",
        "pipelines",
        "searches",
        "service-components",
        "silenced",
        "sumo-logic-metrics-handlers",
        "tcp-stream-handlers"
      ],
      "verbs": [
        "get",
        "list",
        "create",
        "update",
        "delete"
      ]
    }
  ]
}
```

Rules attributes

resources	
description	Types of resources that the rule has permission to access. Read resource types to learn more about available types.
required	true
type	Array
allowed values (roles)	Namespaced resource types and the special resource type `*`.
allowed values (cluster roles)	Namespaced resource types, cluster-wide resource types, and the special resource type `*`.
example	**YML** `resources:` `- checks` `- entities` `- events` `- filters` `- handlers` `- hooks` `- mutators` `- pipelines` `- searches` `- service-components` `- silenced` `- sumo-logic-metrics-handlers` `- tcp-stream-handlers` **JSON** `{` `"resources": [`

```
    "checks",
    "entities",
    "events",
    "filters",
    "handlers",
    "hooks",
    "mutators",
    "pipelines",
    "searches",
    "service-components",
    "silenced",
    "sumo-logic-metrics-handlers",
    "tcp-stream-handlers"
  ]
}
```

resource_names	
description	Names of specific individual resources that the rule has permission to access. Resource name permissions are only taken into account for requests that use `get`, `update`, and `delete` verbs.
required	false
type	Array
example	**YML** `resource_names:` `- check-cpu` **JSON** `{` `  "resource_names": [` `    "check-cpu"` `  ]` `}`

verbs	
description	Type of access the rule will apply.
required	true
type	Array
allowed values	`get`, `list`, `create`, `update`, `delete`
example	**YML** `verbs:` `- get` `- list` `- create` `- update` `- delete` **JSON** `{` `  "verbs": [` `  "get",` `  "list",` `  "create",` `  "update",` `  "delete"` `  ]` `}`

Role binding and cluster role binding specification

Top-level attributes for role binding and cluster role binding resources

api_version

description	Top-level attribute that specifies the Sensu API group and version. For role binding and cluster role binding resources in this version of Sensu, this attribute should always be `core/v2`.
required	Required for role binding and cluster role binding definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

metadata

description	Top-level collection of metadata about the role binding or cluster role binding. The `metadata` map is always at the top level of the role binding or cluster role binding definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for role binding and cluster role binding resources for details. ***NOTE**: Cluster role binding definitions do not include a `namespace` attribute in the resource metadata.*
required	Required for role binding and cluster role binding definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

example	**YML** ```yaml metadata: annotations: managed-by: prod-admin created_by: admin labels: environment: prod1 region: us-west-1 sensu.io/managed_by: sensuctl name: prod-user namespace: production ``` **JSON** ```json { "metadata": { "annotations": { "managed-by": "prod-admin" }, "created_by": "admin", "labels": { "environment": "prod1", "region": "us-west-1", "sensu.io/managed_by": "sensuctl" }, "name": "prod-user", "namespace": "production" } } ```

spec

description	Top-level map that includes the role binding and cluster role binding spec attributes.
required	Required for role binding or cluster role binding definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.

type	Map of key-value pairs

example (role)

YML

```yaml
spec:
  role_ref:
    name: prod-admin
    type: Role
  subjects:
  - name: oncall
    type: Group
  - name: angela
    type: User
```

JSON

```json
{
  "spec": {
    "role_ref": {
      "name": "prod-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "oncall",
        "type": "Group"
      },
      {
        "name": "angela",
        "type": "User"
      }
    ]
  }
}
```

example (cluster role)

YML

```yaml
spec:
  role_ref:
    name: global-event-reader
    type: ClusterRole
```

```
  subjects:
  - name: global-event-readers
    type: Group
  - name: angela
    type: User
```

JSON

```
{
  "spec": {
    "role_ref": {
      "name": "global-event-reader",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "global-event-readers",
        "type": "Group"
      },
      {
        "name": "angela",
        "type": "User"
      }
    ]
  }
}
```

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Role bindings should always be type `RoleBinding`. Cluster role bindings should always be type `ClusterRoleBinding`.
required	Required for role binding and cluster role binding definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

example (role binding)	`type: RoleBinding` **JSON** `{ "type": "RoleBinding" }` **YML**
example (cluster role binding)	`type: ClusterRoleBinding` **JSON** `{ "type": "ClusterRoleBinding" }`

Metadata attributes for role binding and cluster role binding resources

annotations	
description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null` **YML**

example

```
annotations:
  managed-by: prod-admin
```

JSON

```
{
  "annotations": {
    "managed-by": "prod-admin"
  }
}
```

created_by

description	Username of the Sensu user who created or last updated the role binding or cluster role binding. Sensu automatically populates the `created_by` field when the role binding or cluster role binding is created or updated.
required	false
type	String

YML

example

```
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: prod1` `  region: us-west-1` `  sensu.io/managed_by: sensuctl` **JSON** `{` `  "labels": {` `  "environment": "prod1",` `  "region": "us-west-1",` `  "sensu.io/managed_by": "sensuctl"` `  }` `}`

name

description	Unique string used to identify the role binding or cluster role binding. Role binding and cluster role binding names cannot contain special

characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each role binding must have a unique name within its namespace. Each cluster role binding must have a unique name.

required	true
type	String

YML

example

```
name: prod-user
```

JSON

```
{
  "name": "prod-user"
}
```

namespace

description

Sensu RBAC namespace that the role binding belongs to.

> ***NOTE**: Cluster role binding definitions do not include a `namespace` attribute in the resource metadata.*

required	false
type	String
default	`default`

YML

example

```
namespace: production
```

JSON

```
{
```

```
  "namespace": "production"
}
```

Spec attributes for role binding and cluster role binding resources

role_ref	
description	Name and type for the role or cluster role to bind to the users and groups listed in the subjects array. Read role_ref attributes for more information.
required	true
type	Hash
example (role binding)	**YML** `role_ref:` `  name: prod-admin` `  type: Role` **JSON** `{` `  "role_ref": {` `  "name": "prod-admin",` `  "type": "Role"` `  }` `}`
example (cluster role binding)	**YML** `role_ref:` `  name: global-event-reader` `  type: ClusterRole` **JSON**

```
{
  "role_ref": {
    "name": "global-event-reader",
    "type": "ClusterRole"
  }
}
```

subjects	
description	Users and groups to bind with the role or cluster role listed in the role_ref attribute. Read subjects attributes for more information.
required	true
type	Array
example	**YML** `subjects:` `- name: oncall` `  type: Group` `- name: angela` `  type: User` **JSON** `{` `  "subjects": [` `  {` `  "name": "oncall",` `  "type": "Group"` `  },` `  {` `  "name": "angela",` `  "type": "User"` `  }` `  ]` `}`

`role_ref` *attributes*

name

description	Name of the role or cluster role to bind in the role binding or cluster role binding.
required	true
type	String
example	**YML** `name: event-reader` **JSON** `{ "name": "event-reader" }`

type

description	The sensuctl create resource type for the role or cluster role. Use `Role` if you are binding a role. Use `ClusterRole` if you are binding a cluster role.
required	true
type	String
example	**YML** `type: Role` **JSON**

```
{
  "type": "Role"
}
```

`subjects` *attributes*

name	
description	Name of the user resource or group resource to bind in the role binding or cluster role binding.
required	true
type	String
example	**YML** `name: alice` **JSON** `{ "name": "alice" }`
example with prefix	**YML** `name: ad:alice` **JSON** `{ "name": "ad:alice" }`

type	
description	The sensuctl create resource type for the user or group to bind. Use `User` if you are binding a user. Use `Group` if you are binding a group.
required	true
type	String
example	**YML** `type: Group` **JSON** `{ "type": "Group" }`

Maintain Sensu

The Maintain Sensu category includes information to keep your Sensu installation up-to-date and running smoothly.

Upgrade or migrate

Follow the upgrade guide for step-by-step instructions to upgrade to the latest version of Sensu from any earlier version. The upgrade instructions include details about important changes between versions that could affect your upgrade and any special requirements to make sure your upgrade is successful.

If you are still using Sensu Core or Sensu Enterprise, follow Migrate from Sensu Core and Sensu Enterprise to Sensu Go to upgrade to Sensu Go. The migrate guide includes links to Sensu's migration resources and Core and Enterprise configuration translation tools, as well as instructions for installing Sensu Go alongside your existing Sensu Core or Enterprise instance.

Troubleshoot

Use the Sensu troubleshooting guide to diagnose and resolve common problems, and read about tuning options for specific performance issues. Learn how to read, configure, and find the logs produced by Sensu services. Sensu log messages can help you identify and solve backend startup errors and permissions issues.

The troubleshooting guide also describes how to use Sensu handlers and filters to test and debug your observability pipeline and diagnose problems related to dynamic runtime assets.

Manage license

Read the license reference to learn how to activate your commercial license. The license reference also explains how to view your license details and expiration date and find your current entity count and limits.

Upgrade Sensu

To upgrade to the latest version of Sensu Go:

1. Install or upgrade to the latest packages or Docker image.

   > ***NOTE**: If you're upgrading a Sensu cluster, upgrade **all** of your Sensu backends before you run the `sensu-backend upgrade` command in step 5.*

2. For systems that use `systemd`, run:

   ```
   sudo systemctl daemon-reload
   ```

3. Restart the Sensu agent:

   ```
   sudo systemctl restart sensu-agent
   ```

4. Restart the Sensu backend:

   ```
   sudo systemctl restart sensu-backend
   ```

5. Run a single upgrade command on one your Sensu backends to migrate the cluster:

   ```
   sensu-backend upgrade
   ```

   To skip confirmation and immediately run the upgrade command, add the `--skip-confirm` flag:

   ```
   
   ```

```
sensu-backend upgrade --skip-confirm
```

> ***NOTE**: If you are deploying a new Sensu cluster rather than upgrading from a previous version, you do not need to run the `sensu-backend upgrade` command.*

6. Enter `y` or `n` to confirm if you did *not* add the `--skip-confirm` flag in step 5. Otherwise, skip this step.

7. Wait a few seconds for the upgrade command to run. You may notice some inconsistencies in your entity list until the cluster finishes upgrading. Despite this, your cluster will continue to publish standard check requests and process events.

   If you run the upgrade command more than once, it will not harm the cluster — you'll just receive a response that the upgrade command has already been run.

   Some minor versions do not involve database-specific changes, and the `sensu-backend upgrade` tool will report that nothing was upgraded. Check the release notes to confirm whether a version has database-specific changes that require a backend upgrade.

8. Confirm the installed version for the agent, backend, and sensuctl:

```
sensu-agent version
```

```
sensu-backend version
```

```
sensuctl version
```

> ***PRO TIP**: If your upgrade is unsuccessful, read the version-specific information on this page and complete the instructions for each version, starting with your current version and continuing up to the version you want to install.*
>
> *For example, to debug an upgrade from 5.5.0 to 6.4.0, start with Upgrade Sensu clusters from 5.7.0 or earlier to 5.8.0 or later.*

Upgrade to Sensu Go 6.5.0 from any previous version

To use pipelines, you must upgrade your Sensu agents to Sensu Go 6.5.0. Agents that are not upgraded to 6.5.0 will run checks, send observability events to the backend, and use the handlers that are defined in check handlers arrays, but they will not run pipelines.

Upgrade to Sensu Go 6.4.0 from any previous version

In Sensu Go 6.4.0, we upgraded the embedded etcd version from 3.3.22 to 3.5.0. As a result, for deployments that use embedded etcd, 6.4.0 is not backward-compatible with previous versions of the Sensu backend. In addition, Sensu 6.4.0 is not backward-compatible for PostgreSQL deployments.

> ***NOTE**: Sensu Go 6.4.0 is backward-compatible for deployments that use external etcd, as well as with previous versions of the Sensu agent.*

For embedded etcd deployments, before you upgrade to Sensu Go 6.4.0, use the etcd snapshot and restore process to create a full etcd database backup. If you use PostgreSQL, make sure to back up your PostgreSQL database also.

If you make a full etcd database backup (and a PostgreSQL database backup, if you use PostgreSQL) before upgrading to 6.4.0, you will be able to restore your pre-6.4.0 deployment if you need to revert to an earlier Sensu release.

After creating a full backup of your embedded etcd database and your PostgreSQL database, if you use PostgreSQL, you can complete the upgrade process.

CommonName deprecation in Go 1.15

Sensu Go 6.4.0 upgrades the Go version from 1.13.15 to 1.16.5. As of Go 1.15, certificates must include their CommonName (CN) as a Subject Alternative Name (SAN) field.

To prevent connection errors after upgrading to Sensu Go 6.4.0, follow Generate certificates to make sure your certificates' SAN fields include their CNs.

Upgrade to Sensu Go 6.2.0 from any previous version

Prior to Sensu Go 6.0, sensu-backend allowed you to delete a namespace even when other resources still referenced that namespace. As of Sensu Go 6.0, it is not possible to delete namespaces that are referenced in other resources. As a result, users whose configuration predates Sensu Go 6.0 may have lingering resources, including check configurations, that reference non-existent namespaces.

Upgrading to Sensu Go 6.2.0 requires sensu-backend to upgrade check configurations. If you have check configurations that reference non-existent namespaces, the 6.2.0 upgrade operation will fail when it encounters one of these check configurations. You will receive an error message like this:

```
{"component":"store-providers","error":"the namespace test does not
exist","level":"error","msg":"error enabling round robin scheduling,backend restart
required","time":"2020-12-27T08:41:59Z"}
```

When this happens, the backend is effectively halted and subsequent restarts will result in the same state.

Remove checks that reference non-existent namespaces

Use the following commands with the jq utility to identify and remove checks that reference deleted namespaces before you upgrade to Sensu Go 6.2.0.

> ***NOTE**: If you have already upgraded to Sensu Go 6.2.0, you can work around this issue by temporarily reverting your Sensu instance to Sensu Go 6.1.4. Then, recreate the missing namespaces referenced in your check configurations and upgrade again to 6.2.0.*

These commands use a Sensu backend running on `localhost` in the example URL and the environment variable $SENSU_API_KEY to represent a valid API key.

1. Get a list of existing namespaces. Run:

   ```
   curl -s -H "Authorization: Key $SENSU_API_KEY"
   http://localhost:8080/api/core/v2/namespaces | jq '[.[].name]'
   ```

   In this example, the existing namespaces are `stage` and `dev`.

```
[
   "stage",
   "dev"
]
```

2. Print the name and namespace for any checks that reference a namespace that is not specified in the jq expression on the same line (in this example, `["stage","dev"]`):

```
curl -s -H "Authorization: Key $SENSU_API_KEY"
http://localhost:8080/api/core/v2/checks | jq '["stage","dev"] as $valid | .[]
| select(.metadata.namespace as $in | $valid | index($in) | not) | {name:
.metadata.name, namespace: .metadata.namespace}'
```

   Your jq expression should include all of the namespaces you retrieved in step 1 (`stage` and `dev`).

```
{
  "name": "check-cpu",
  "namespace": "test"
}
```

3. Recreate the missing `test` namespace so you can delete the `check-cpu` check.

```
sensuctl namespace create test
```

4. Delete the `check-cpu` check.

```
sensuctl check delete check-cpu --namespace test
```

5. Delete the `test` namespace, which is now empty after you deleted `check-cpu` in step 4.

```
sensuctl namespace delete test
```

After completing these commands, you can upgrade to 6.2.0.

Upgrade to Sensu Go 6.1.0 from 6.0.0

If you are using 6.0.0 and have a large number of events in PostgreSQL, you may experience a short period of unavailability after you upgrade to 6.1.0. This pause will occur while the optimized selector information is populating during automatic database migration. It may last for a period of a few seconds to a few minutes.

This pause may extend to API request processing, so sensuctl and the web UI may also be unavailable during the migration.

Upgrade to Sensu Go 6.0 from a 5.x deployment

Before you upgrade to Sensu 6.0, use `sensuctl dump` to create a backup of your existing installation.

You will not be able to downgrade to a Sensu 5.x version after you upgrade your database to Sensu 6.0 after you restart the backend in the upgrade process.

Upgrade to Sensu Go 5.16.0 from any earlier version

As of Sensu Go 5.16.0, Sensu's free entity limit is 100 entities. All commercial features are available for free in the packaged Sensu Go distribution for up to 100 entities.

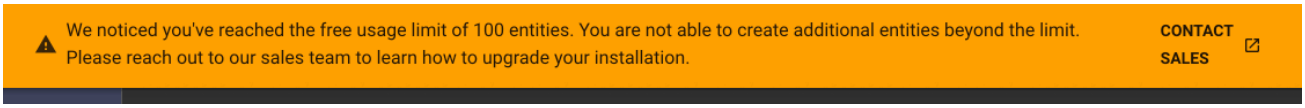
When you upgrade to 5.16.0, if your existing unlicensed instance has more than 100 entities, Sensu will continue to monitor those entities. However, if you try to create any new entities via the HTTP API or sensuctl, you will receive the following message:

```
This functionality requires a valid Sensu Go license with a sufficient entity limit. To get a valid license file, arrange a trial, or increase your entity limit, contact Sales.
```

Connections from new agents will fail and result in a log message like this:

```
{"component":"agent","error":"handshake failed with status
402","level":"error","msg":"reconnection attempt failed","time":"2019-11-
20T05:49:24-07:00"}
```

In the web UI, you will receive the following message when you reach the 100-entity limit:



If your Sensu instance includes more than 100 entities, contact Sales to learn how to upgrade your installation and increase your limit.Read our blog announcement for more information about our usage policy.

Upgrade Sensu clusters from 5.7.0 or earlier to 5.8.0 or later

> ***NOTE**: This section applies only to Sensu clusters with multiple backend nodes.*

Due to updates to etcd serialization, you must shut down Sensu clusters with multiple backend nodes while upgrading from Sensu Go 5.7.0 or earlier to 5.8.0 or later.Read the backend reference for more information about stopping and starting backends.

Upgrade Sensu backend binaries to 5.1.0

> ***NOTE**: This section applies only to Sensu backend binaries downloaded from `s3-us-west-2.amazonaws.com/sensu.io/sensu-go`, not to Sensu RPM or DEB packages.*

For Sensu backend binaries, the default `state-dir` in 5.1.0 is now `/var/lib/sensu/sensu-backend` instead of `/var/lib/sensu`.To upgrade your Sensu backend binary to 5.1.0, first download the latest version.Then, make sure the `/etc/sensu/backend.yml` configuration file specifies a `state-dir`.To continue using `/var/lib/sensu` as the `state-dir` to store backend data, add the following configuration to `/etc/sensu/backend.yml`:

```
state-dir: "/var/lib/sensu"
```

Then restart the backend:

```
sudo systemctl restart sensu-backend
```

Migrate from Sensu Core and Sensu Enterprise to Sensu Go

This guide includes general information for migrating your Sensu instance from Sensu Core and Sensu Enterprise to Sensu Go. For instructions and tools to help you translate your Sensu configuration from Sensu Core and Enterprise to Sensu Go, review the Sensu Translator project.

> ***NOTE**: The information in this guide applies to Sensu Enterprise as well as Sensu Core, although we refer to "Sensu Core" for brevity.*
>
> *The step for translating integrations, contact routing, and LDAP authentication applies to Sensu Enterprise (but **not** Sensu Core), and it is designated as Sensu Enterprise-only.*

Sensu Go includes important changes to all parts of Sensu: architecture, installation, resource definitions, the observation data (event) model, check dependencies, filter evaluation, and more. Sensu Go also includes many powerful commercial features to make monitoring easier to build, scale, and offer as a self-service tool to your internal customers.

Sensu Go is available for Debian- and RHEL-family distributions and Docker. The Sensu Go agent is also available for Windows.

> ***WARNING**: To install Sensu Go alongside your current Sensu instance, you must upgrade to at least Sensu Core 1.9.0-2. If you need to upgrade, contact Sensu.*

Aside from this migration guide, these resources can help you migrate from Sensu Core or Sensu Enterprise to Sensu Go:

- **Sensu Community Slack**: Join hundreds of other Sensu users in our Community Slack, where you can ask questions and benefit from tips others picked up during their own Sensu Go migrations.
- **Sensu Community Forum**: Drop a question in our dedicated category for migrating to Go.
- **Sensu Go workshop**: Download the workshop environment and try out some monitoring workflows with Sensu Go.
- **Sensu Translator**: Use this command-line tool to generate Sensu Go configurations from your

Sensu Core config files.

We also offer **commercial support** and **professional services** packages to help with your Sensu Go migration.

Configuration management with Ansible, Chef, and Puppet

Configuration management integrations for Sensu Go are available for Ansible, Chef, and Puppet:

- Ansible collection for Sensu Go and documentation site
- Chef cookbook for Sensu Go — contact us for more information
- Puppet module for Sensu Go

Packaging

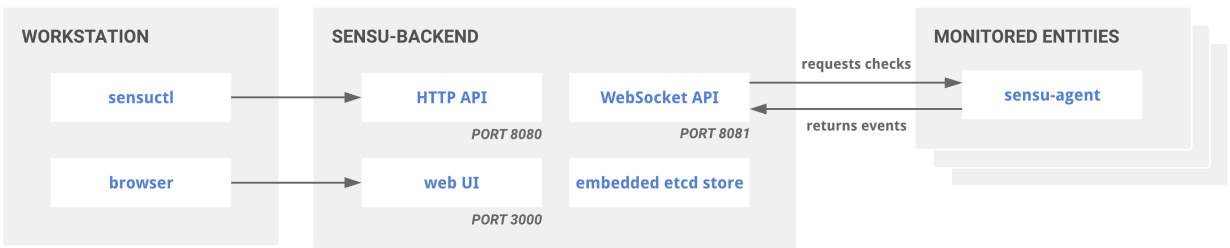
Sensu Go is provided as three packages: sensu-go-backend, sensu-go-agent, and sensu-go-cli (sensuctl). This is a fundamental change in Sensu terminology from Sensu Core: in Sensu Go, the server is now the backend.

Clients are represented within Sensu Go as abstract entities that can describe a wider range of system components such as network gear, a web server, or a cloud resource.

Read Sensu concepts and terminology to learn more about new terms in Sensu Go.

Architecture

In Sensu Go, an embedded transport and etcd datastore replace the external RabbitMQ transport and Redis datastore in Sensu Core.



Single Sensu Go backend or standalone architecture

In Sensu Go, the Sensu backend and agent are configured with YAML files or the `sensu-backend` or `sensu-agent` command line tools rather than JSON files. Sensu checks and pipeline elements are configured via the API or sensuctl tool in Sensu Go instead of JSON files.

The **Sensu backend** is powered by an embedded transport and etcd datastore and gives you flexible, automated workflows to route metrics and alerts. Sensu backends require persistent storage for their embedded database, disk space for local dynamic runtime asset caching, and several exposed ports:

- 2379 (gRPC) Sensu storage client: Required for Sensu backends using an external etcd instance
- 2380 (gRPC) Sensu storage peer: Required for etcd cluster members to communicate directly with their peers
- 3000 (HTTP/HTTPS) Sensu web UI: Required for all Sensu backends using a Sensu web UI
- 8080 (HTTP/HTTPS) Sensu API: Required for all users accessing the Sensu API
- 8081 (WS/WSS) Agent API: Required for all Sensu agents connecting to a Sensu backend

Sensu agents are lightweight clients that run on the infrastructure components you want to monitor. Agents automatically register with Sensu as entities and are responsible for creating check and metric events to send to the backend event pipeline.

The Sensu agent uses:

- 3030 (TCP/UDP) Sensu agent socket: Required for Sensu agents using the agent socket
- 3031 (HTTP) Sensu agent API: Required for all users accessing the agent API
- 8125 (UDP) StatsD listener: Required for all Sensu agents using the StatsD listener

The agent TCP and UDP sockets are deprecated in favor of the agent API.

Agents that use Sensu dynamic runtime assets require some disk space for a local cache.

Read the backend, agent, and sensuctl reference docs for more information.

Entities

Clients are represented within Sensu Go as abstract entities that can describe a wide range of system

components such as network gear, a web server, or a cloud resource.

Sensu Go includes agent entities that run a Sensu agent and the familiar proxy entities. Sensu Go also includes service entities, which represent business services in the business service monitoring (BSM) feature.

Read the entities reference and the guide to monitoring external resources for more information about Sensu Go entities.

Checks

In Sensu Go, checks work with Sensu agents to produce observability events automatically. The Sensu backend coordinates check execution by comparing the subscriptions specified in check and entity definitions to determine which entities should receive execution requests for a given check.

Subdue

Sensu Go checks include a `subdues` attribute that allows you to set specific periods of time when Sensu will not execute the check. Read Subdues in the checks reference for more information and examples.

You can also use cron scheduling in Sensu Go checks to specify when checks **should** be executed.

Standalone checks

Sensu Go does not include standalone checks. Read Self-service monitoring checks in Sensu Go to learn more about using role-based access control (RBAC), dynamic runtime assets, and entity subscriptions to achieve similar functionality to Sensu Core's standalone checks in Sensu Go.

Check hooks

Check hooks are a distinct resource type in Sensu Go, which allows you to create, manage, and reuse hooks independently of check definitions. You can also execute multiple hooks for any given response code.

Default handler

Sensu Go does not try to run a default handler when executing checks whose definitions do not specify a handler name. In Sensu Go, you explicitly add the name of a handler in a pipeline and reference the pipeline in your check definition.

Events

In Sensu Go, all check results are considered events and are processed by pipelines, which include event filters, mutators, and handlers.

Use Sensu Go's built-in is_incident filter to recreate the Sensu Core behavior in which only check results with a non-zero status are considered events.

Handlers

Sensu Go includes pipe and TCP/UDP handlers, but not transport handlers. To create similar functionality to transport handlers in Sensu Go, create a pipe handler that connects to a message bus and injects event data into a queue.

Sensu Go also includes streaming handlers, such as the Sumo Logic metrics handler, to provide persistent connections for transmitting Sensu observation data to remote data storage services to help prevent data bottlenecks.

Filters

In Sensu Go, JavaScript expressions replace the Ruby eval logic in Sensu Core, opening up powerful ways to filter events based on occurrences and other event attributes. As a result, Sensu Go does not include the built-in occurrence-based event filter in Sensu Core. To replicate the Sensu Core occurrence-based filter's functionality, use Sensu Go's repeated events filter definition.

Sensu Go includes three built-in event filters: is_incident, not_silenced, and has_metrics. Sensu Go does not include a built-in check dependencies filter, but you can use the sensu/sensu-dependencies-filter dynamic runtime asset to replicate the built-in check dependencies filter functionality from Sensu Core.

Sensu Go event filters do not include the `when` event filter attribute. Use Sensu query expressions to build custom functions that provide granular control of time-based filter expressions.

Fatigue check filter

The sensu/sensu-go-fatigue-check-filter dynamic runtime asset is a JavaScript implementation of the `occurrences` filter from Sensu Core. This filter looks for check and entity annotations in each event it receives and uses the values of those annotations to configure the filter's behavior on a per-event basis.

The Sensu Translator version 1.1.0 retrieves occurrence and refresh values from a Sensu Core check definition and outputs them as annotations in a Sensu Go check definition, compatible with the fatigue check filter.

However, the Sensu Translator doesn't automatically add the sensu/sensu-go-fatigue-check-filter dynamic runtime asset or the filter configuration you need to run it. To use the sensu/sensu-go-fatigue-check-filter dynamic runtime asset, you must register it, create a correctly configured event filter definition, and add the event filter to the list of filters on applicable handlers.

Dynamic runtime assets

The `sensu-install` tool in Sensu Core is replaced by dynamic runtime assets in Sensu Go.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

You can still install Sensu Community plugins in Ruby via `sensu-install` by installing sensu-plugins-ruby.Read Install plugins for more information.

Role-based access control (RBAC)

Role-based access control (RBAC) is a built-in feature of the open-source version of Sensu Go.RBAC allows you to manage and access users and resources based on namespaces, groups, roles, and bindings. To set up RBAC in Sensu Go, read the RBAC reference and Create a read-only user.

Silencing

Silencing is disabled by default in Sensu Go. You must explicitly enable silencing by creating silencing resource definitions with sensuctl, the Sensu web UI, or core/v2/silenced API endpoints.Read the Sensu Go silencing reference for more information.

Token substitution

The syntax for token substitution changed to double curly braces in Sensu Go (from triple colons in Sensu Core).

Aggregates

Sensu Go supports check aggregates with the sensu/sensu-aggregate-check dynamic runtime asset.

API

In addition to the changes to resource definitions, Sensu Go includes new versioned APIs. Read the API overview for more information.

Step-by-step migration instructions

Step 1: Install Sensu Go

1. Install the Sensu Go backend

The Sensu backend is available for Debian- and RHEL-family distributions and Docker. Read the installation guide to install, configure, and start the Sensu backend according to your deployment strategy.

2. Log in to the Sensu web UI

The Sensu Go web UI provides a unified view of your observability events with user-friendly tools to reduce alert fatigue and manage your Sensu instance. After starting the Sensu backend, open the web UI by visiting http://localhost:3000. You may need to replace `localhost` with the hostname or IP address where the Sensu backend is running.

To log in, enter your Sensu user credentials or use Sensu's default admin credentials (username: `admin` and password: `P@ssw0rd!`).

3. Install sensuctl on your workstation

Sensuctl is a command line tool for managing resources within Sensu. It works by calling Sensu's HTTP API to create, read, update, and delete resources, events, and entities. Sensuctl is available for Linux, Windows, and macOS. Read the installation guide to install and configure sensuctl.

4. Set up Sensu users

Use Sensu's built-in RBAC to manage and access users and resources based on namespaces, groups, roles, and bindings. To set up RBAC in Sensu Go, read the RBAC reference and Create a read-only user.

In Sensu Go, namespaces partition resources within a Sensu instance. Sensu Go entities, checks, handlers, and other namespaced resources belong to a single namespace. The Sensu translator places all translated resources into the `default` namespace — we'll use the translater in a moment.

In addition to built-in RBAC, Sensu Go's commercial features include support for authentication using Microsoft Active Directory (AD) and standards-compliant Lightweight Directory Access Protocol tools like OpenLDAP.

5. Install agents

The Sensu agent is available for Debian- and RHEL-family distributions, Windows, and Docker. Read the installation guide to install, configure, and start Sensu agents.

If you're doing a side-by-side migration, add `api-port` (default: `3031`) and `socket-port` (default: `3030`) to your agent configuration (`/etc/sensu/agent.yml` or `C:\ProgramData\sensu\config\agent.yml.example`). This prevents the Sensu Go agent API and socket from conflicting with the Sensu Core client API and socket.

```
api-port: 3031
socket-port: 3030
```

You can also disable these features in the agent configuration using the `disable-socket` and `disable-api` configuration options.

Sensu should now be installed and functional. The next step is to translate your Sensu Core configuration to Sensu Go.

Step 2: Translate your configuration

Use the Sensu Translator command line tool to transfer your Sensu Core checks, handlers, and mutators to Sensu Go.

1. Run the translator

Install dependencies:

```
yum install -q -y rubygems ruby-devel
```

Install the Sensu translator:

```
gem install sensu-translator
```

Run the Sensu translator to translate all configuration in /etc/sensu/conf.d to Sensu Go and output to /sensu_config_translated:

```
sensu-translator -d /etc/sensu/conf.d -o /sensu_config_translated
```

As an option, you can also translate your configuration in sections according to resource type.

If translation is successful, you should receive a few callouts followed by `DONE!`, similar to this example:

```
Sensu 1.x filter translation is not yet supported
Unable to translate Sensu 1.x filter: only_production {:attributes=>{:check=>
{:environment=>"production"}}}
DONE!
```

Combine your config into a sensuctl-readable format.

***NOTE**: for use with `sensuctl create`, do not use a comma between resource objects in Sensu Go resource definitions in JSON format.*

```
find sensu_config_translated/ -name '*.json' -exec cat {} \; >
sensu_config_translated_singlefile.json
```

Most attributes are ready to use as-is, but you'll need to adjust your Sensu Go configuration manually to migrate some of Sensu's features.

***NOTE**: To streamline a comparison of your Sensu Core configuration with your Sensu Go configuration, output your current Sensu Core configuration using the API: `curl -s http://127.0.0.1:4567/settings | jq . > sensu_config_original.json`.*

2. Translate checks

Review your Sensu Core check configuration for the following attributes, and make the corresponding updates to your Sensu Go configuration.

Core attribute	Manual updates required in Sensu Go config
`::: foo :::`	Update the syntax for token substitution from triple colons to double curly braces. For example: `{{ foo }}`
`stdin: true`	No updates required. Sensu Go checks accept data on stdin by default.
`handlers: default`	Sensu Go does not have a default handler. Create a handler named `default` to continue using this pattern.
`subdues`	Check subdues are not available in Sensu Go.
`standalone: true`	Standalone checks are not supported in Sensu Go, although you can achieve similar functionality using role-based access control, dynamic runtime assets, and entity subscriptions. The translator assigns all Core standalone checks to a `standalone` subscription in Sensu Go. Configure one or more Sensu Go agents with the `standalone` subscription to execute Sensu Core standalone checks.
`metrics: true`	Review the translate metric checks section.

`proxy_requests`	Review the translate proxy requests section.
`subscribers: roundrobin...`	Remove `roundrobin` from the subscription name, and add the `round_robin` check attribute set to `true`.
`aggregate`	Check aggregates are supported through the sensu/sensu-aggregate-check.
`hooks`	Review the translate hooks section.
`dependencies`	Use the sensu/sensu-dependencies-filter dynamic runtime asset.

***PRO TIP**: When using token substitution in Sensu Go and accessing labels or annotations that include `.`, like `sensu.io.json_attributes`, use the `index` function. For example, `{{index .annotations "web_url"}}` substitutes the value of the `web_url` annotation; `{{index .annotations "production.ID"}}` substitutes the value of the `production.ID` annotation.*

Translate metric checks

The Sensu Core `type: metric` attribute is not part of the Sensu Go check spec, so you'll need to adjust it manually. Sensu Core checks could be configured as `type: metric`, which told Sensu to always handle the check regardless of the check status output. This allowed Sensu Core to process output metrics via a handler even when the check status was not in an alerting state.

Sensu Go treats output metrics as first-class objects, so you can process check status as well as output metrics via different event pipelines. Read the guide to metric output to update your metric checks with the `output_metric_handlers` and `output_metric_format` attributes and use `output_metric_tags` to enrich extracted metrics output.

Translate proxy requests and proxy entities

Read Monitor external resources to re-configure `proxy_requests` attributes and update your proxy check configuration. Read the entities reference to re-create your proxy client configurations as Sensu Go proxy entities.

Translate hooks

Check hooks are now a resource type in Sensu Go, so you can create, manage, and reuse hooks independently of check definitions. You can also execute multiple hooks for any given response code. Read the guide and hooks reference docs to re-create your Sensu Core hooks as Sensu Go hook resources.

Custom attributes

Instead of custom check attributes, Sensu Go allows you to add custom labels and annotations to entities, checks, dynamic runtime assets, hooks, filters, mutators, handlers, and silences. Review the metadata attributes section in the reference documentation for more information about using labels and annotations (for example, metadata attributes for entities).

The Sensu Translator stores all check extended attributes in the check metadata annotation named `sensu.io.json_attributes`. Read the checks reference for more information about using labels and annotations in check definitions.

3. Translate event filters

Ruby eval logic used in Sensu Core filters is replaced with JavaScript expressions in Sensu Go, opening up powerful possibilities to combine filters with filter dynamic runtime assets. As a result, you'll need to rewrite your Sensu Core filters in Sensu Go format.

First, review your Core handlers to identify which filters are being used. Then, follow the event filters reference and guide to using filters to re-write your filters using Sensu Go expressions and event data. Check out the blog post on filters for a deep dive into Sensu Go filter capabilities.

Sensu Core hourly filter:

```
{
  "filters": {
    "recurrences": {
      "attributes": {
        "occurrences": "eval: value == 1 || value % 60 == 0"
      }
    }
  }
}
```

Sensu Go hourly filter:

YML

```
---
type: EventFilter
```

```
api_version: core/v2
metadata:
  name: hourly
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0
  runtime_assets: null
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "hourly"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0"
    ],
    "runtime_assets": null
  }
}
```

4. Translate handlers

In Sensu Go, all check results are considered events and are processed by event handlers. Use the built-in is_incident filter to recreate the Sensu Core behavior, in which only check results with a non-zero status are considered events.

> ***NOTE**: Silencing is disabled by default in Sensu Go and must be explicitly enabled. Read the silencing reference to create silences in Sensu Go.*

Review your Sensu Core check configuration for the following attributes and make the corresponding updates to your Sensu Go configuration.

Core attribute	Manual updates required in Sensu Go config
`filters: occurrences`	Replicate the built-in occurrences filter in Sensu Core with the sensu/sensu-go-fatigue-check-filter.
`type: transport`	Achieve similar functionailty to transport handlers in Sensu Core with a Sensu Go pipe handler that connects to a message bus and injects event data into a queue.
`filters: check_dependencies`	Use the sensu/sensu-dependencies-filter dynamic runtime asset.
`severities`	Sensu Go does not support severities.
`handle_silenced`	Silencing is disabled by default in Sensu Go and must be explicitly enabled using sensuctl, the web UI, or core/v2/silenced API endpoints.
`handle_flapping`	All check results are considered events in Sensu Go and are processed by pipelines.

5. Upload your config to your Sensu Go instance

After you review your translated configuration, make any necessary updates, and add resource definitions for any filters and entities you want to migrate, you can upload your Sensu Go config using sensuctl.

```
sensuctl create --file /path/to/config.json
```

***PRO TIP**: `sensuctl create` (and `sensuctl delete`) are powerful tools to help you manage your Sensu configs across namespaces. Read the sensuctl reference for more information.*

Access your Sensu Go config using the Sensu API.

Set up a local API testing environment by saving your Sensu credentials and token as environment variables. This command requires curl and jq.

```
export SENSU_USER=admin && SENSU_PASS=P@ssw0rd!
export SENSU_TOKEN=`curl -XGET -u "$SENSU_USER:$SENSU_PASS" -s
http://localhost:8080/auth | jq -r ".access_token"`
```

Return a list of all configured checks:

```
curl -H "Authorization: Bearer $SENSU_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

Return a list of all configured handlers:

```
curl -H "Authorization: Bearer $SENSU_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers
```

You can also access your Sensu Go configuration in JSON or YAML using sensuctl.For example, `sensuctl check list --format wrapped-json`.Run `sensuctl help` To view available commands For more information about sensuctl's output formats (`json`, `wrapped-json`, and `yaml`), read the sensuctl reference.

Step 3: Translate plugins and register dynamic runtime assets

Sensu plugins

Within the Sensu Plugins org, review individual plugin READMEs for compatibility status with Sensu Go.For handler and mutators plugins, review the Sensu plugins README to map event data to the Sensu Go event format.This allows you to use Sensu plugins for handlers and mutators with Sensu Go without re-writing them.

To re-install Sensu plugins onto your Sensu Go agent nodes (check plugins) and backend nodes (mutator and handler plugins), read the guide to installing the `sensu-install` tool for use with Sensu Go.

Sensu Go dynamic runtime assets

The `sensu-install` tool in Sensu Core is replaced by dynamic runtime assets in Sensu Go.Dynamic

runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Although dynamic runtime assets are not required to run Sensu Go, we recommend using assets to install plugins where possible. You can still install Sensu Community plugins in Ruby via `sensu-install` by installing sensu-plugins-ruby. Read Install plugins for more information.

Sensu supports dynamic runtime assets for checks, filters, mutators, and handlers. Discover, download, and share dynamic runtime assets with Bonsai, the Sensu asset hub.

To create your own dynamic runtime assets, read the asset reference and guide to sharing an asset on Bonsai. To contribute to converting a Sensu plugin to a dynamic runtime asset, read Contributing Asset for Existing Ruby Sensu Plugins at the Sensu Community Forum on Discourse.

Step 4: Translate Sensu Enterprise-only features

Integrations

Most Sensu Enterprise integrations as available as Sensu Go assets. Read the guide to installing plugins with assets to register assets with Sensu and update your Sensu Go handler definitions.

- Chef
- Email
- Graphite
- InfluxDB
- IRC
- Jira
- PagerDuty
- ServiceNow
- Slack
- VictorOps

Contact routing

Contact routing is available in Sensu Go woth the sensu/sensu-go-has-contact-filter dynamic runtime asset. Read Route alerts with event filters to set up contact routing in Sensu Go.

LDAP

In addition to built-in RBAC, Sensu includes license-activated support for authentication using Microsoft Active Directory and standards-compliant Lightweight Directory Access Protocol tools like OpenLDAP.

Step 5: Sunset your Sensu Core instance

When you're ready to sunset your Sensu Core instance, stop the Sensu Core services according to the instructions for your platform — these instructions are listed under **Operating Sensu** on each platform's page.

After you stop the Sensu Core services, follow package removal instructions for your platform to uninstall the Sensu Core package.

Tune Sensu

This page describes tuning options that may help restore proper operation if you experience performance issues with your Sensu installation.

> ***NOTE**: Before you tune your Sensu installation, read Troubleshoot Sensu, Hardware requirements, and Deployment architecture for Sensu. These pages describe common problems and solutions, planning and optimization considerations, and other recommendations that may resolve your issue without tuning adjustments.*

Latency tolerances for etcd

If you use embedded etcd for storage, you might notice high network or storage latency.

To make etcd more latency-tolerant, increase the values for the `etcd election timeout` and `etcd heartbeat interval` backend configuration options. For example, you might increase `etcd-election-timeout` from 3000 to 5000 and `etcd-heartbeat-interval` from 300 to 500.

Read the etcd tuning documentation for etcd-specific tuning best practices.

Advanced backend configuration options for etcd

The backend reference describes other advanced configuration options in addition to etcd election timeout and heartbeat interval.

Adjust these values with caution. Improper adjustment can increase memory and CPU usage or result ir a non-functioning Sensu instance.

Input/output operations per second (IOPS)

The speed with which write operations can be completed is important to Sensu cluster performance and health. Make sure to provision Sensu backend infrastructure to provide sustained input/output

operations per second (IOPS) appropriate for the rate of observability events the system will be required to process.

Read Backend recommended configuration and Hardware sizing for details.

PostgreSQL settings

The datastore reference lists the PostgreSQL configuration parameters and settings we recommend as a starting point for your `postgresql.conf` file. Adjust the parameters and settings as needed based on your hardware and performance observations.

Read the PostgreSQL parameters documentation for information about setting parameters.

Agent reconnection rate

> ***COMMERCIAL FEATURE**: Access the `agent-rate-limit` backend configuration option in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

It may take several minutes for all agents to reconnect after a sensu-backend restart, especially if you have a large number of agents. The agent reconnection rate depends on deployment variables like the number of CPUs, disk space, network speeds, whether you're using a load balancer, and even physical distance between agents and backends.

Although many variables affect the agent reconnection rate, a reasonable estimate is approximately 100 agents per backend per second. If you observe slower agent reconnection rates in your Sensu deployment, consider using the `agent-rate-limit` backend configuration option.

The `agent-rate-limit` backend configuration option allows you to set the maximum number of agent transport WebSocket connections per second, per backend. Set the agent-rate-limit to 100 to improve agent reconnection rate and reduce the time required for all of your agents to reconnect after a backend restart.

Splay and proxy check scheduling

Adjust the `splay` and `splay_coverage` check attributes to tune proxy check executions across an interval. Read Fine-tune proxy check scheduling with splay for an example.

Tokens and resource re-use

Tokens are placeholders in a check, hook, or dynamic runtime asset definition that the agent replaces with entity information before execution. You can use tokens to fine-tune check, hook, and asset attributes on a per-entity level while reusing resource definitions.

Read the tokens reference for token syntax and examples.

Occurrences and alert fatigue

Use the `occurrences` and `occurrences_watermark` event attributes in event filters to tune incident notifications and reduce alert fatigue.

Troubleshoot Sensu

Service logging

Logs produced by Sensu services (sensu-backend and sensu-agent) are often the best place to start when troubleshooting a variety of issues.

Log file locations

Linux

Sensu services print structured log messages to standard output. To capture these log messages to disk or another logging facility, Sensu services use capabilities provided by the underlying operating system's service management. For example, logs are sent to the journald when systemd is the service manager, whereas log messages are redirected to `/var/log/sensu` when running under sysv init schemes. If you are running systemd as your service manager and would rather have logs written to `/var/log/sensu/`, read forwarding logs from journald to syslog.

For journald targets, use these commands to follow the logs. Replace the `<service>` variable with the name of the desired service (for example, `backend` or `agent`).

SHELL

```
journalctl --follow --unit sensu-<service>
```

SHELL

```
journalctl --follow --unit sensu-<service>
```

SHELL

```
journalctl --follow --unit sensu-<service>
```

For log file targets, use these commands to follow the logs.Replace the `<service>` variable with the name of the desired service (for example, `backend` or `agent`).

SHELL

```
tail --follow /var/log/sensu/sensu-<service>
```

SHELL

```
tail --follow /var/log/sensu/sensu-<service>
```

SHELL

```
tail --follow /var/log/sensu/sensu-<service>
```

***NOTE**: Platform versions are listed for reference only and do not supersede the documented supported platforms.*

Narrow your search to a specific timeframe

Use the `journald` keyword `since` to refine the basic `journalctl` commands and narrow your search by timeframe.

Retrieve all the logs for sensu-backend since yesterday:

```
journalctl -u sensu-backend --since yesterday | tee sensu-backend-$(date +%Y-%m-%d).log
```

Retrieve all the logs for sensu-agent since a specific time:

```
journalctl -u sensu-agent --since 09:00 --until "1 hour ago" | tee sensu-agent-$(date +%Y-%m-%d).log
```

Retrieve all the logs for sensu-backend for a specific date range:

```
journalctl -u sensu-backend --since "2015-01-10" --until "2015-01-11 03:00" | tee sensu-backend-$(date +%Y-%m-%d).log
```

Logging edge cases

If a Sensu service experiences a panic crash, the service may seem to start and stop without producing any output in journalctl. This is due to a bug in systemd.

In these cases, try using the `_COMM` variable instead of the `-u` flag to access additional log entries:

```
journalctl _COMM=sensu-backend.service --since yesterday
```

Windows

The Sensu agent stores service logs to the location specified by the `log-file` configuration option (default `%ALLUSERSPROFILE%\sensu\log\sensu-agent.log`, `C:\ProgramData\sensu\log\sensu-agent.log` on standard Windows installations). For more information about managing the Sensu agent for Windows, read the agent reference. You can also view agent events using the Windows Event Viewer, under Windows Logs, as events with source SensuAgent.

If you're running a binary-only distribution of the Sensu agent for Windows, you can follow the service log printed to standard output using this command:

```
Get-Content -  Path "C:\scripts\test.txt" -Wait
```

Log levels

Each log message is associated with a log level that indicates the relative severity of the event being logged:

Log level	Description
panic	Severe errors that cause the service to shut down in an unexpected state

fatal	Fatal errors that cause the service to shut down (status 0)
error	Non-fatal service error messages
warn	Warning messages that indicate potential issues
info	Information messages that represent service actions
debug	Detailed service operation messages to help troubleshoot issues
trace	Confirmation messages about whether a rule authorized a request

You can configure these log levels by specifying the desired log level as the value of `log-level` in the service configuration file (`agent.yml` or `backend.yml`) or as an argument to the `--log-level` command line flag:

```
sensu-agent start --log-level debug
```

You must restart the service after you change log levels via configuration files or command line arguments.For help with restarting a service, read the agent reference or backend reference.

Increment log level verbosity

To increment the log level verbosity at runtime for the backend, run:

```
kill -s SIGUSR1 $(pidof sensu-backend)
```

To increment the log level verbosity at runtime for the agent, run:

```
kill -s SIGUSR1 $(pidof sensu-agent)
```

When you increment the log at the trace level (the most verbose log level), the log will wrap around to the error level.

Sensu backend startup errors

The following errors are expected when starting up a Sensu backend with the default configuration:

```
{"component":"etcd","level":"warning","msg":"simple token is not cryptographically signed","pkg":"auth","time":"2019-11-04T10:26:31-05:00"}
{"component":"etcd","level":"warning","msg":"set the initial cluster version to 3.5","pkg":"etcdserver/membership","time":"2019-11-04T10:26:31-05:00"}
{"component":"etcd","level":"warning","msg":"serving insecure client requests on 127.0.0.1:2379, this is strongly discouraged!","pkg":"embed","time":"2019-11-04T10:26:33-05:00"}
```

The `serving insecure client requests` warning is an expected warning from the embedded etcd database. TLS configuration is recommended but not required. For more information, read the etcd security documentation.

CommonName deprecation in Go 1.15

As of Go 1.15, certificates must include their CommonName (CN) as a Subject Alternative Name (SAN) field.

The following logged error indicates that a certificate used to secure Sensu does not include the CN as a SAN field:

```
{"component":"agent","error":"x509: certificate relies on legacy Common Name field, use SANs or temporarily enable Common Name matching with GODEBUG=x509ignoreCN=0","level":"error","msg":"reconnection attempt failed","time":"2021-06-29T11:07:51+02:00"}
```

If you see this connection error, follow Generate certificates to make sure your certificates' SAN fields include their CNs.

Permission issues

The Sensu user and group must own files and folders within `/var/cache/sensu/` and

`/var/lib/sensu/`. You will receive a logged error like those listed here if there is a permission issue with either the sensu-backend or the sensu-agent:

```
{"component":"agent","error":"open /var/cache/sensu/sensu-agent/assets.db:
permission denied","level":"fatal","msg":"error executing sensu-agent","time":"2019-
02-21T22:01:04Z"}
{"component":"backend","level":"fatal","msg":"error starting etcd: mkdir
/var/lib/sensu: permission denied","time":"2019-03-05T20:24:01Z"}
```

Use a recursive `chown` to resolve permission issues with the sensu-backend:

```
sudo chown -R sensu:sensu /var/cache/sensu/sensu-backend
```

or the sensu-agent:

```
sudo chown -R sensu:sensu /var/cache/sensu/sensu-agent
```

Handlers and event filters

Whether implementing new workflows or modifying existing workflows, you may need to troubleshoot various stages of the event pipeline.

Create an agent API test event

In many cases, generating events using the agent API will save you time and effort over modifying existing check configurations.

Here's an example that uses cURL with the API of a local sensu-agent process to generate test-event check results:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
```

```
  "check": {
    "metadata": {
      "name": "test-event"
    },
    "status": 2,
    "output": "this is a test event targeting the email_ops handler",
    "handlers": [ "email_ops" ]
  }
}' \
http://127.0.0.1:3031/events
```

Use a debug handler

It may also be helpful to review the complete event object being passed to your workflows. We recommend using a debug handler like this one to write an event to disk as JSON data:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: debug
spec:
  type: pipe
  command: cat > /var/log/sensu/debug-event.json
  timeout: 2
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "debug"
  },
  "spec": {
    "type": "pipe",
    "command": "cat > /var/log/sensu/debug-event.json",
    "timeout": 2
```

```
  }
}
```

With this handler definition installed in your Sensu backend, you can add the `debug` to the list of handlers in your test event:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "test-event"
    },
    "status": 2,
    "output": "this is a test event targeting the email_ops handler",
    "handlers": [ "email_ops", "debug" ]
  }
}' \
http://127.0.0.1:3031/events
```

The observability event data should be written to `/var/log/sensu/debug-event.json` for inspection. The contents of this file will be overwritten by every event sent to the `debug` handler.

***NOTE**: When multiple Sensu backends are configured in a cluster, event processing is distributed across all members. You may need to check the filesystem of each Sensu backend to locate the debug output for your test event.*

Manually execute a handler

If you are not receiving events via a handler even though a check is generating events as expected, follow these steps to manually execute the handler and confirm whether the handler is working properly.

1. List all events:

```
```

```
sensuctl event list
```

Choose an event from the list to use for troubleshooting and note the event's check and entity names.

2. Navigate to the `/var/cache/sensu/sensu-backend/` directory:

```
cd /var/cache/sensu/sensu-backend/
```

3. Run `ls` to list the contents of the `/var/cache/sensu/sensu-backend/` directory. In the list, identify the handler's dynamic runtime asset SHA.

> ***NOTE**: If the list includes more than one SHA, run `sensuctl asset list`. In the response, the Hash column contains the first seven characters for each asset build's SHA. Note the hash for your build of the handler asset and compare it with the SHAs listed in the `/var/cache/sensu/sensu-backend/` directory to find the correct handler asset SHA.*

4. Navigate to the `bin` directory for the handler asset SHA. Before you run the command below, replace `<handler_asset_sha>` with the SHA you identified in the previous step.

```
cd <handler_asset_sha>/bin
```

5. Run the command to manually execute the handler. Before you run the command below, replace the following text:

   - `<entity_name>`: Replace with the entity name for the event you are using to troubleshoot.
   - `<check_name>`: Replace with the check name for the event you are using to troubleshoot.
   - `<handler_command>`: Replace with the `command` value for the handler you are troubleshooting.

```
sensuctl event info <entity_name> <check_name> --format json |
./<handler_command>
```

If your handler is working properly, you will receive an alert for the event via the handler. The response for your manual execution command will also include a message to confirm notification was sent. In this case, your Sensu pipeline is not causing the problem with missing events.

If you do not receive an alert for the event, the handler is not working properly. In this case, the manual execution response will include the message `Error executing <handler_asset_name>:` followed by a description of the specific error to help you correct the problem.

Dynamic runtime assets

Use the information in this section to troubleshoot error messages related to dynamic runtime assets.

Incorrect asset filter

Dynamic runtime asset filters allow you to scope an asset to a particular operating system or architecture. For an example, read the asset reference. An improperly applied asset filter can prevent the asset from being downloaded by the desired entity and result in error messages both on the agent and the backend illustrating that the command was not found:

Agent log entry

```
{
  "asset": "check-disk-space",
  "component": "asset-manager",
  "entity": "sensu-centos",
  "filters": [
    "true == false"
  ],
  "level": "debug",
  "msg": "entity not filtered, not installing asset",
  "time": "2020-09-12T18:28:05Z"
}
```

Backend event

YML

```
---
timestamp: 1568148292
check:
  command: check-disk-space
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  publish: true
  runtime_assets:
  - sensu-disk-checks
  subscriptions:
  - caching_servers
  proxy_entity_name: ''
  check_hooks:
  stdin: false
  subdue:
  ttl: 0
  timeout: 0
  round_robin: false
  duration: 0.001795508
  executed: 1568148292
  history:
  - status: 127
    executed: 1568148092
  issued: 1568148292
  output: 'sh: check-disk-space: command not found'
  state: failing
  status: 127
  total_state_change: 0
  last_ok: 0
  occurrences: 645
  occurrences_watermark: 645
  output_metric_format: ''
  output_metric_handlers:
  output_metric_tags:
  env_vars:
  metadata:
    name: failing-disk-check
    namespace: default
metadata:
  namespace: default
```

JSON


```
{
  "timestamp": 1568148292,
  "check": {
    "command": "check-disk-space",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "sensu-disk-checks"
    ],
    "subscriptions": [
      "caching_servers"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 0.001795508,
    "executed": 1568148292,
    "history": [
      {
        "status": 127,
        "executed": 1568148092
      }
    ],
    "issued": 1568148292,
    "output": "sh: check-disk-space: command not found\n",
    "state": "failing",
    "status": 127,
    "total_state_change": 0,
    "last_ok": 0,
    "occurrences": 645,
    "occurrences_watermark": 645,
    "output_metric_format": "",
```

```
    "output_metric_handlers": null,
    "output_metric_tags": null,
    "env_vars": null,
    "metadata": {
      "name": "failing-disk-check",
      "namespace": "default"
    }
  },
  "metadata": {
    "namespace": "default"
  }
}
```

If you receive a message like this, review your asset definition — it means that the entity wasn't able to download the required asset due to asset filter restrictions. To review the filters for an asset, use the sensuctl `asset info` command with a `--format` flag:

SHELL

```
sensuctl asset info sensu-disk-checks --format yaml
```

SHELL

```
sensuctl asset info sensu-disk-checks --format wrapped-json
```

Conflating operating systems with families

A common asset filter issue is conflating operating systems with the family they're a part of. For example, although Ubuntu is part of the Debian family of Linux distributions, Ubuntu is not the same as Debian. A practical example might be:

YML

```
filters:
- entity.system.platform == 'debian'
- entity.system.arch == 'amd64'
```

JSON

```
{
  "filters": [
    "entity.system.platform == 'debian'",
    "entity.system.arch == 'amd64'"
  ]
}
```

This would not allow an Ubuntu system to run the asset.

Instead, the asset filter should look like this:

YML

```
filters:
- entity.system.platform_family == 'debian'
- entity.system.arch == 'amd64'
```

JSON

```
{
  "filters": [
    "entity.system.platform_family == 'debian'",
    "entity.system.arch == 'amd64'"
  ]
}
```

or

YML

```
filters:
- entity.system.platform == 'ubuntu'
- entity.system.arch == 'amd64'
```

JSON

```
{
  "filters": [
```

```
    "entity.system.platform == 'ubuntu'",
    "entity.system.arch == 'amd64'"
  ]
}
```

This would allow the asset to be downloaded onto the target entity.

Running the agent on an unsupported Linux platform

If you run the Sensu agent on an unsupported Linux platform, the agent might fail to correctly identify your version of Linux and could download the wrong version of an asset.

This issue affects Linux distributions that do not include the `lsb_release` package in their default installations. In this case, `gopsutil` may try to open `/etc/lsb_release` or try to run `/usr/bin/lsb_release` to find system information, including Linux version. Since the `lsb_release` package is not installed, the agent will not be able to discover the Linux version as expected.

To resolve this problem, install the [`lsb_release` package]() for your Linux distribution.

Checksum mismatch

When a downloaded dynamic runtime asset checksum does not match the checksum specified in the asset definition, the agent logs a message similar to this example:

```
{
  "asset": "check-disk-space",
  "component": "asset-manager",
  "entity": "sensu-centos",
  "filters": [
    "true == false"
  ],
  "level": "debug",
  "msg": "error getting assets for check: could not validate downloaded asset
$ASSET_NAME (X.X MB): sha512 of downloaded asset
(6b73p32XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX) does not match specified sha512 in
asset definition
(e6b7c8eXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX)",
  "time": "2019-09-12T18:28:05Z"
}
```

To correct this issue, first confirm that the URL in the asset definition is valid. Manually download the asset with a cURL or wget command and make sure that the downloaded file is a valid `tar.gz` file with the contents you expect.

If the downloaded `tar.gz` file contents are correct, use the `sha512sum` command to calculate the asset checksum and manually confirm that the checksum in the downloaded asset definition is correct.

If the checksum in the downloaded asset definition is correct, confirm that there is enough space available in `/tmp` to download the asset. On Linux systems, the Sensu agent downloads assets into `/tmp`. The log error message specifies the size of the asset artifact in parentheses after the asset name. If space in `/tmp` is insufficient, asset downloads will be truncated and the checksum will not be validated.

Certificate error when fetching assets

When Sensu cannot fetch the assets referenced in a resource definition, the agent logs a message similar to this example:

```
error getting assets for check: error fetching asset: Get
"https://assets.bonsai.sensu.io/2940de675113d07710c0f896efa8b43b7c301c5c/sensu-
plugins-process-checks_4.0.0_centos_linux_amd64.tar.gz": x509: certificate signed by
unknown authority
```

To correct this issue, confirm that you can download the asset from one of the agent hosts using the link quoted in the error message. If the download does not work, the problem may be due to a proxy between the agent and the internet or the proxy settings.

If there are no proxies or no proxy settings of concern, you may need to update the certificate store on your agents. The https://assets.bonsai.sensu.io SSL certificate uses the AWS Private Certificate Authority (CA), which your agents' operating systems should be configured to trust.

If you are using PowerShell, you may see this error if PowerShell is configured to use TLS 1.0 — https://assets.bonsai.sensu.io requires TLS 1.2.

To check which TLS version PowerShell is using, run:

```
[Net.ServicePointManager]::SecurityProtocol
```

If the response does not include `Tls12`, run the following command to require it:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
```

Etcd clusters

Some issues require you to investigate the state of the etcd cluster or data stored within etcd. In these cases, we suggest using the etcdctl tool to query and manage the etcd database.

Sensu's supported packages do not include the etcdctl executable, so you must get it from a compatible etcd release.

Configure etcdctl environment variables

To use etcdctl to investigate etcd cluster and data storage issues, first run these commands to configure etcdctl environment variables:

```
export ETCDCTL_API=3
export ETCDCTL_CACERT=/etc/sensu/ca.pem
export
ETCDCTL_ENDPOINTS="https://backend01:2379,https://backend02:2379,https://backend03:2
379"
```

If your etcd uses client certificate authentication, run these commands too:

```
export ETCDCTL_CERT=/etc/sensu/cert.pem
export ETCDCTL_KEY=/etc/sensu/key.pem
```

View cluster status and alarms

Use the commands listed here to retrieve etcd cluster status and list and clear alarms.

To retrieve etcd cluster status:

```
etcdctl endpoint status
```

To retrieve a list of etcd alarms:

```
etcdctl alarm list
```

To clear etcd alarms:

```
etcdctl alarm disarm
```

Restore a cluster with an oversized database

The default Sensu backend configuration sets the maximum etcd database size to 4 GB.If you suspect your etcd database exceeds 4 GB, run this command to confirm cluster size:

```
etcdctl endpoint status
```

The response will list the current cluster status and database size:

```
https://backend01:2379, 88db026f7feb72b4, 3.5.0, 4.1GB, false, 144, 18619245
https://backend02:2379, e98ad7a888d16bd6, 3.5.0, 4.1GB, true, 144, 18619245
https://backend03:2379, bc4e39432cbb36d, 3.5.0, 4.1GB, false, 144, 18619245
```

To restore an etcd cluster with a database size that exceeds 4 GB:

1. Get the current revision number:

```
etcdctl endpoint status --write-out="json" | egrep -o '"revision":[0-9]*' |
egrep -o '[0-9].*'
```

2. Compact to revision and substitute the current revision for `$rev`:

```
etcdctl compact $rev
```

3. Defragment to free up space:

```
etcdctl defrag
```

4. Confirm that the cluster is restored:

```
etcdctl endpoint status
```

   The response should list the current cluster status and database size:

```
https://backend01:2379, 88db026f7feb72b4, 3.5.0, 1.0 MB, false, 144, 18619245
https://backend02:2379, e98ad7a888d16bd6, 3.5.0, 1.0 MB, true, 144, 18619245
https://backend03:2379, bc4e39432cbb36d, 3.5.0, 1.0 MB, false, 144, 18619245
```

Remove and redeploy a cluster

***PRO TIP**: If you are using external etcd, use etcd snapshots to keep a backup so that you can restore your Sensu resources if you have to redeploy your cluster. For extra reassurance, take regular etcd snapshots and make regular backups with sensuctl dump in addition to etcd's running snapshots.*

If you are using embedded etcd, use sensuctl dump to make regular backups.

If you wait until cluster nodes are failing, it may not be possible to make a backup. For example, in a three-node cluster, if one node fails, you will still be able to make a backup. If two nodes fail, the whole cluster will be down and you will not be able to create a snapshot or run sensuctl dump.

You may need to completely remove a cluster and redeploy it in cases such as:

- Failure to reach consensus after losing more than `(N-1)/2` cluster members
- Etcd configuration issues
- Etcd corruption, perhaps from disk filling
- Unrecoverable hardware failure

To remove and redeploy a cluster:

1. Open a terminal window for each cluster member.
2. Stop each cluster member backend:

   ```
   systemctl stop sensu-backend
   ```

3. Confirm that each backend stopped:

   ```
   systemctl status sensu-backend
   ```

   For each backend, the response should begin with the following lines:

   ```
   ● sensu-backend.service - The Sensu Backend service.
   Loaded: loaded (/usr/lib/systemd/system/sensu-backend.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
   ```

4. Delete the etcd directories for each cluster member:

   ```
   rm -rf /var/lib/sensu/sensu-backend/etcd/
   ```

5. Follow the Sensu backend configuration instructions to reconfigure a new cluster.

6. Initialize a backend to specify admin credentials:

```
sensu-backend init --interactive
```

When you receive prompts for your username and password, replace `<YOUR_USERNAME>` and `<YOUR_PASSWORD>` with the administrator username and password you want to use for the cluster members:

```
Admin Username: <YOUR_USERNAME>
Admin Password: <YOUR_PASSWORD>
```

7. Restore your cluster from a snapshot or backup:

    - Follow the etcd restore process (for external etcd).
    - Use sensuctl create (for external or embedded etcd).

Datastore performance

In a default deployment, Sensu uses etcd datastore for both configuration and state. As the number of checks and entities in your Sensu installation increases, so does the volume of read and write requests to etcd database.

One trade-off in etcd's design is its sensitivity to disk and CPU latency. When certain latency tolerances are regularly exceeded, failures will cascade. Sensu will attempt to recover from these conditions when it can, but this may not be successful.

To maximize Sensu Go performance, we recommend that you:

- Follow our recommended backend hardware configuration.
- Implement documented etcd system tuning practices.
- Benchmark your etcd storage volume to establish baseline IOPS for your system.
-

- Scale event storage using PostgreSQL with round robin scheduling enabled to reduce the overall volume of etcd transactions.

As your Sensu deployments grow, preventing issues associated with poor datastore performance relies on ongoing collection and review of Sensu time-series performance metrics.

Symptoms of poor performance

At the Sensu backend's default "warn" log level, you may receive messages like these from your backend:

```
{"component":"etcd","level":"warning","msg":"read-only range request
\"key:\\\"/sensu.io/handlers/default/keepalive\\\" limit:1 \" with result
\"range_response_count:0 size:6\" took too long (169.767546ms) to
execute","pkg":"etcdserver","time":"..."}
```

The above message indicates that a database query ("read-only range request") exceeded a 100-millisecond threshold hard-coded into etcd. Messages like these are helpful because they can alert you to a trend, but these occasional warnings don't necessarily indicate a problem with Sensu. For example, you may receive this message if you provision attached storage but do not mount it to the etcd data directory.

However, a trend of increasingly long-running database transactions will eventually lead to decreased reliability. You may experience symptoms of these conditions as inconsistent check execution behavior or configuration updates that are not applied as expected.

As the etcd tuning documentation states:

> An etcd cluster is very sensitive to disk latencies. Since etcd must persist proposals to its log, disk activity from other processes may cause long fsync latencies. […] etcd may miss heartbeats, causing request timeouts and temporary leader loss.

When Sensu's etcd component doesn't recieve sufficient CPU cycles or its file system can't sustain a sufficient number of IOPS, transactions will begin to timeout, leading to cascading failures.

A message like this indicates that syncing the etcd database to disk exceeded another threshold:

```
{"component":"etcd","level":"warning","msg":"sync duration of 1.031759056s, expected
```

```
less than 1s","pkg":"wal","time":"..."}
```

These subsequent “retrying of unary invoker failed” messages indicate failing requests to etcd:

```
{"level":"warn","ts":"...","caller":"clientv3/retry_interceptor.go:62","msg":"retryi
ng of unary invoker failed","target":"endpoint://client-6f6bfc7e-cf31-4498-a564-
78d6b7b3a44e/localhost:2379","attempt":0,"error":"rpc error: code = Canceled desc =
context canceled"}
```

On busy systems you may also receive output like “message repeated 5 times” indicating that failing requests were retried multiple times.

In many cases, the backend service detects and attempts to recover from errors like these, so you may receive a message like this:

```
{"component":"backend","error":"error from keepalived: internal error: etcdserver:
request timed out","level":"error","msg":"backend stopped working and is
restarting","time":"..."}
```

This may result in a crash loop that is difficult to recover from. You may observe that the Sensu backend process continues running but is not listening for connections on the agent WebSocket, API, or web UI ports. The backend will stop listening on those ports when the etcd database is unavailable.

Check execution errors

The Sensu backend sends check requests to all matching subscriptions. If an entity and a check have multiple matching subscriptions, the entity will receive a separate check request for each matching subscription. The entity could receive both check requests almost simultaneously.

As a result, you may see one or more of the following error messages:

```
{"component":"agent","error":"check execution still in progress:
<CHECK_NAME>","level":"error","msg":"error handling message","time":"..."}

{"component":"agent","error":"check request is older than a previously received
```

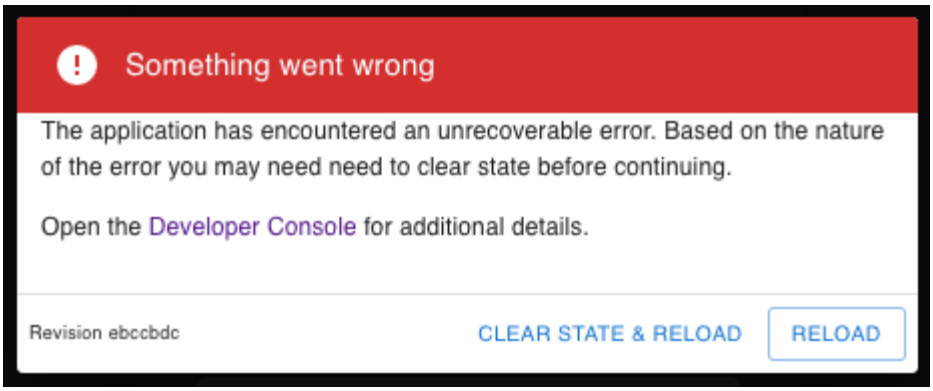
```
check request","level":"error","msg":"error handling message","time":"..."}

{"component":"agent","warning":"check request has already been received - agent and check may have multiple matching subscriptions","level":"warn","msg":"error handling message","time":"..."}
```

Entities may execute the duplicate check requests quickly enough to prevent these errors. In these cases, check `history` and features that rely on it, like flap detection, may behave in unexpected ways.

If you see any of the check execution errors listed above, review the check subscriptions against your entities for multiple matching subscriptions. To prevent the problem, make sure that your checks and entities share only a single subscription.

Web UI errors

If the web UI experiences an error, you may see the following message in the web UI:



The error message indicates something unexpected happened, such as the server failing to return the correct response. Clicking **RELOAD** can resolve most common problems.

More rarely, the error can result from issues like a corrupt cache or a bad persistent state. In these cases, clicking **CLEAR STATE & RELOAD** usually resolves the issue.

Investigate a web UI error

To get more information about a web UI error, open your web browser's developer console to view the error messages your browser logged.

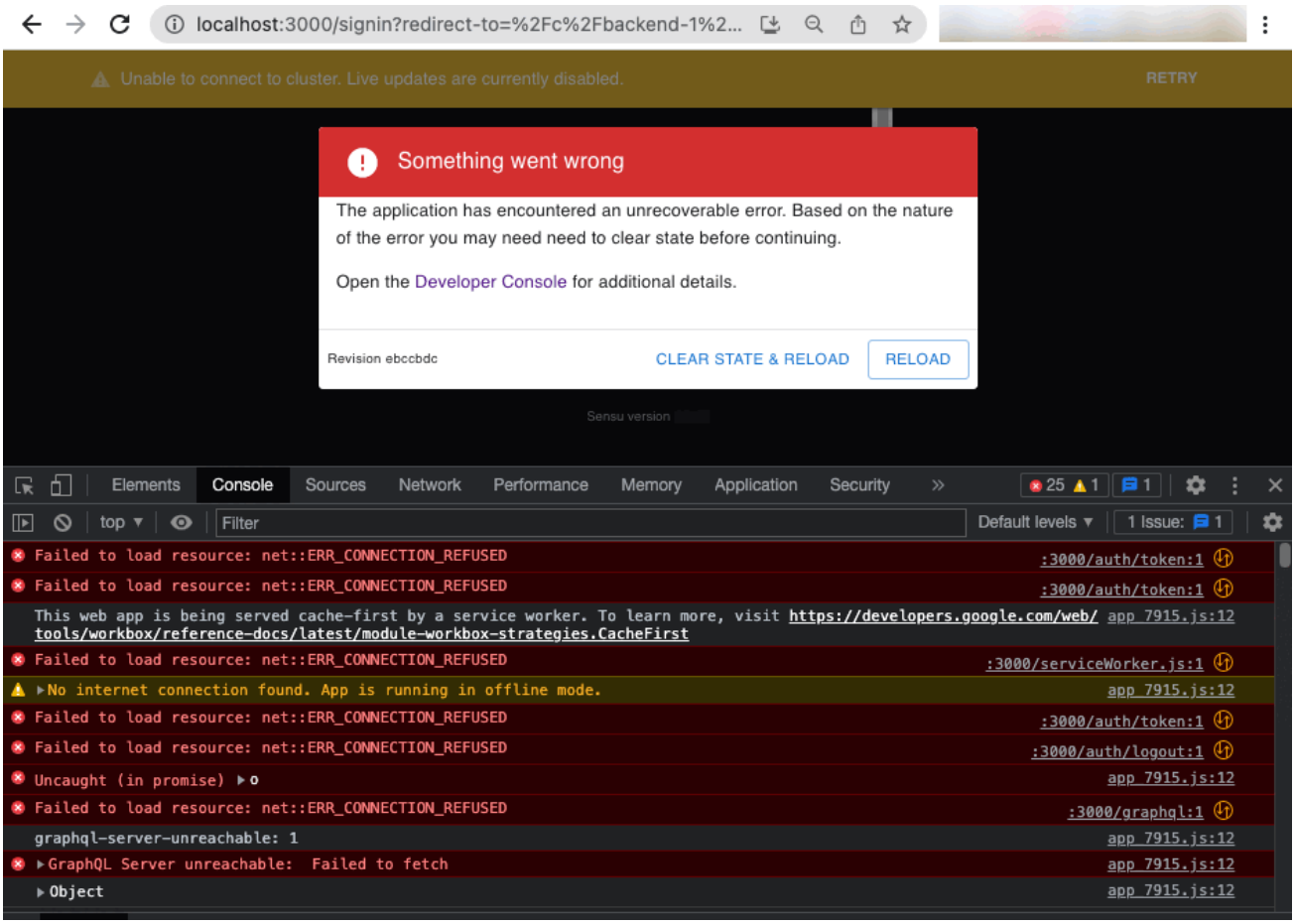
Use these keyboard shortcuts to open the developer console on different operating systems:

Operating system	Keyboard shortcut
Linux	Press `Control` + `Shift` + `J`
macOS	Press `Command` + `Option` + `J`
Windows	Press `Control` + `Shift` + `J`

You can also open the developer console from the browser's menu:

Browser	Menu path
Chrome	Click the ⋮ menu icon, then `More Tools` > `Developer Tools`
Edge	`Tools` > `Developer` > `JavaScript Console`
Firefox	Click the ☰ menu icon, then `More Tools` > `Developer Console`
Safari	`Develop` > `Show JavaScript Console` If you do not see the `Develop` option, open `Safari` > `Preferences` > `Advanced` and select the checkbox for `Show Develop menu in menu bar`

Depending on your browser, the developer console may open in a separate browser window or within the current browser window as shown in this example:



The developer console lists all errors for the current page. Click an error for more information about it.

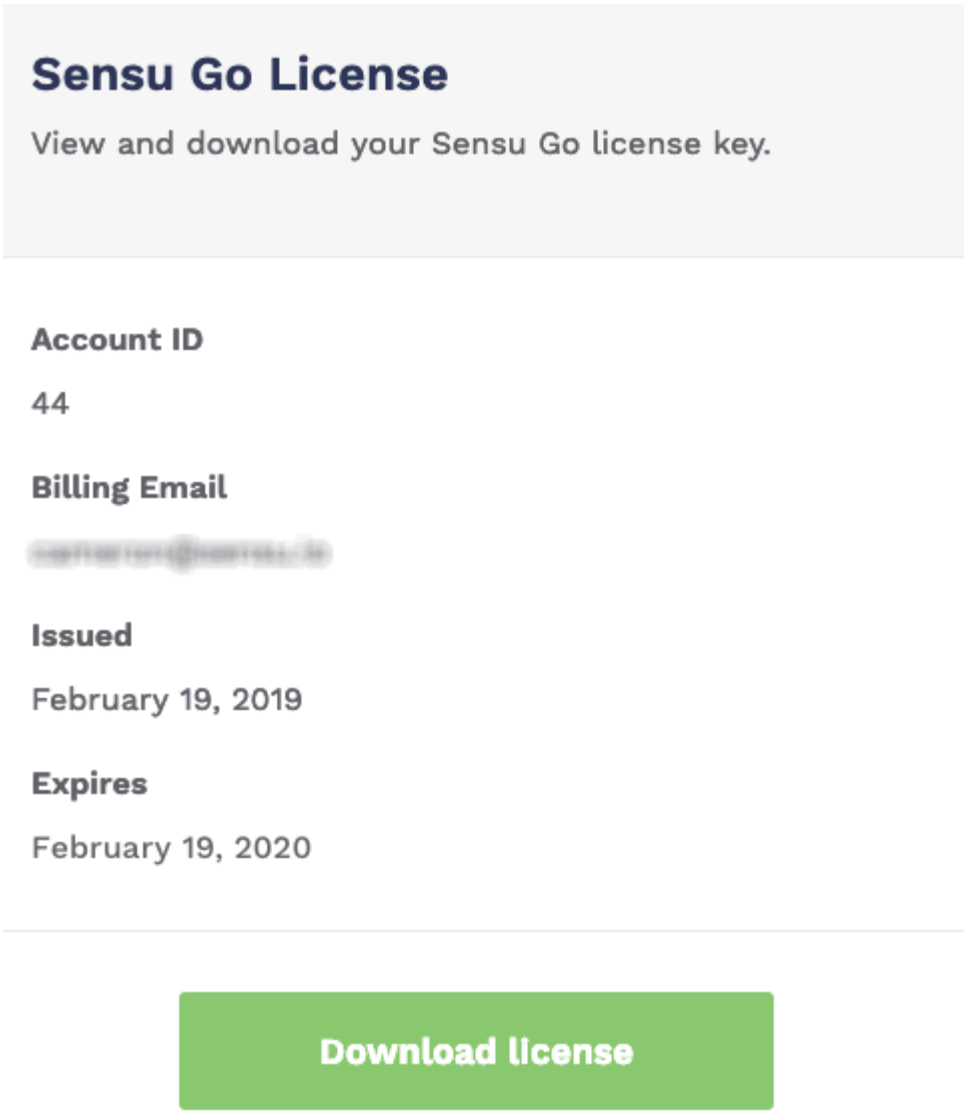
The developer console is part of web developer tools that are included in all modern browsers. These tools may have different names in different browsers (for example, DevTools in Chrome and Developer Tools in Firefox), but they offer similar features. Read the documentation for your browser to learn more about the web developer tools your browser provides.

License reference

Activate your commercial license

If you haven't already, install the backend, agent, and sensuctl and configure sensuctl.

Log in to your Sensu account at account.sensu.io and click **Download license** to download your license file.



Save your license to a file such as `sensu_license.yml` or `sensu_license.json`. With the license file downloaded and saved to a file, you can activate your license with sensuctl or the /license API.

***NOTE**: For clustered configurations, you only need to activate your license for one of the backends within the cluster.*

To activate your license with sensuctl:

SHELL

```
sensuctl create --file sensu_license.yml
```

SHELL

```
sensuctl create --file sensu_license.json
```

Use sensuctl to view your license details at any time:

```
sensuctl license info
```

For an active license, the response should be similar to this example:

```
=== You are currently using 10/100 total entities, 5/50 agent entities, and 5/50
proxy entities
Account Name: Training Team - Sensu
Account ID:   123
Plan:         managed
Version:      1
Features:     all
Issuer:       Sensu, Inc.
Issued:       2020-02-15 15:01:44 -0500 -0500
Valid:        true
Valid Until:  2021-02-15 00:00:00 -0800 -0800
```

This response means you do not have an active license:

```
Error: not found
```

Entity limit

Your commercial license may include the entity limit and entity class limits tied to your Sensu licensing package. Contact Sensu to upgrade your commercial license.

Your Sensu license may include two types of entity limits:

- Entity limit: the maximum number of entities of all classes your license includes. Both agent and proxy entities count toward the overall entity limit.
- Entity class limits: the maximum number of a specific class of entities (for example, agent or proxy) that your license includes.

For example, if your license has an entity limit of 10,000 and an agent entity class limit of 3,000, you cannot run more than 10,000 entities (agent and proxy) total. At the same time, you cannot run more than 3,000 agents. If you use only 1,500 agent entities, you can have 8,500 proxy entities before you reach the overall entity limit of 10,000.

If you have permission to create or update licenses, you will see messages in sensuctl and the web UI when you approach your licensed entity limit. The formula for calculating the threshold for this warning message is `0.03 * entity limit / 1000 + 0.9`. For example, if your entity limit is 1600, the warning threshold is 0.948.

You will also see a warning when you exceed your entity or entity class limit.

View entity count and entity limit

Your current entity count and entity limit are included in the `sensuctl license info` response.

In tabular format, the entity count and limit are included in the response title. To view license info in tabular format, run:

```
sensuctl license info --format tabular
```

The response in tabular format should be similar to this example:

```
=== You are currently using 10/100 total entities, 5/50 agent entities, and 5/50
proxy entities
Account Name: Training Team - Sensu
Account ID:   123
Plan:         managed
Version:      1
Features:     all
Issuer:       Sensu, Inc.
Issued:       2020-02-15 15:01:44 -0500 -0500
Valid:        true
Valid Until:  2021-02-15 00:00:00 -0800 -0800
```

If you have an unlimited entity count, the `sensuctl license info` response title will still include a current count for each type of entity you are using.For example:

```
=== You are currently using 10/unlimited total entities, 5/unlimited agent entities,
and 5/unlimited proxy entities
```

To view license details in YAML or JSON, run:

SHELL

```
sensuctl license info --format yaml
```

SHELL

```
sensuctl license info --format wrapped-json
```

In YAML and JSON formats, the entity count and limit are included as labels:

TEXT

```
---
type: LicenseFile
api_version: licensing/v2
metadata:
```

```
  labels:
    sensu.io/entity-count: "10"
    sensu.io/entity-limit: "100"
spec:
  license:
    version: 1
    issue: Sensu, Inc.
    accountName: Training Team - Sensu
[...]
```

TEXT

```
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issue": "Sensu, Inc.",
      "accountName": "Training Team - Sensu"
    },
    "...": "..."
  }
}
```

You can also find your current entity count and limit in the response headers for any `/api/core` or `/api/enterprise` API request. For example:

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/entities -v -H
"Authorization: Key $SENSU_API_KEY"
```

The response headers will include your current entity count and limit:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Entity-Count: 10
Sensu-Entity-Limit: 100
```

License expiration

To view your commercial license expiration date, log in to your Sensu account.

When your license is within 30 days of expiration, Sensu issues regular warnings in the Sensu backend logs. Users with permission to create or update licenses can also view license expiration information in the web UI by pressing `CTRL .` to open the system information modal.

If your license expires, you will still have access to commercial features, but your entity limit will drop back down to the free limit of 100.

Quick links

- Log in to your Sensu account
- Use the license management API
- Contact Sensu support
- Contact Sensu sales

Monitor Sensu

Use the guides and references in the Monitor Sensu category to successfully monitor your Sensu installation.

Log Sensu services and monitor with Sensu

Learn how to log Sensu services with systemd, including adding log forwarding from journald to syslog, using rsyslog to write logging data to disk, and setting up log rotation.

Read Monitor Sensu with Sensu to monitor the Sensu backend with another Sensu backend or cluster: use a secondary Sensu instance to notify you when your primary Sensu instance is down (and vice versa).

Retrieve cluster health data

The health reference explains how to use Sensu's /health API to ensure your backend is up and running and check the health of your etcd cluster members and PostgreSQL datastore resources. Learn how to read the JSON response for /health API requests by reviewing examples of responses for clusters with healthy and unhealthy members and the response specification.

Learn about Tessen

The Tessen reference explains the Sensu call-home service, which is enabled by default on Sensu backends and required for licensed Sensu instances. We rely on anonymized Tessen data to understand how Sensu is being used and make informed decisions about product improvements.

Log Sensu services with systemd

By default, systems where systemd is the service manager do not write logs to `/var/log/sensu/` for the `sensu-agent` and the `sensu-backend` services. This guide explains how to add log forwarding from journald to syslog, have rsyslog write logging data to disk, and set up log rotation of the newly created log files.

Configure journald

To configure journald to forward logging data to syslog, modify `/etc/systemd/journald.conf` to include the following line:

```
ForwardToSyslog=yes
```

Configure rsyslog

Next, set up rsyslog to write the logging data received from journald to `/var/log/sensu/servicename.log`. In this example, the `sensu-backend` and `sensu-agent` logging data is sent to individual files named after the service. The `sensu-backend` is not required if you're only setting up log forwarding for the `sensu-agent` service.

***NOTE**: Use a `conf` file name that will ensure the file is loaded before the default file in `/etc/rsyslog.d/`, which uses 50. This example uses `40-sensu-backend.conf` and `40-sensu-agent.conf` for this reason.*

1. For the sensu-backend service, in /etc/rsyslog.d/40-sensu-backend.conf, add:

```
if $programname == 'sensu-backend' then {
        /var/log/sensu/sensu-backend.log
        & stop
}
```

2. For the sensu-agent service, in /etc/rsyslog.d/40-sensu-agent.conf, add:

```
if $programname == 'sensu-agent' then {
        /var/log/sensu/sensu-agent.log
        & stop
}
```

3. **On Ubuntu systems**, run `chown -R syslog:adm /var/log/sensu` so syslog can write to that directory.

4. Restart journald:

```
systemctl restart systemd-journald
```

5. Restart rsyslog to apply the new configuration:

```
systemctl restart rsyslog
```

***NOTE**: Sensu log messages include the Sensu log level as part of the log data. Users with rsyslog expertise may be able to extract the log level from Sensu log messages and use rsyslog processing capabilities to separate the log messages into different files based on log level.*

Set up log rotation

Set up log rotation for newly created log files to ensure logging does not fill up your disk.

These examples rotate the log files `/var/log/sensu/sensu-agent.log` and `/var/log/sensu/sensu-backend.log` weekly, unless the size of 100M is reached first. The last seven rotated logs are kept and compressed, with the exception of the most recent log. After rotation, `rsyslog` is restarted to ensure logging is written to a new file and not the most recent rotated file.

1. In /etc/logrotate.d/sensu-agent.conf, add:

```
/var/log/sensu/sensu-agent.log {
    daily
    rotate 7
    size 100M
    compress
    delaycompress
    postrotate
      /bin/systemctl restart rsyslog
    endscript
}
```

2. In /etc/logrotate.d/sensu-backend.conf, add:

```
/var/log/sensu/sensu-backend.log {
    daily
    rotate 7
    size 100M
    compress
    delaycompress
    postrotate
      /bin/systemctl restart rsyslog
    endscript
}
```

Use the following command to find out what logrotate would do if it were executed now based on the above schedule and size threshold. The `-d` flag will output details, but it will not take action on the logs or execute the postrotate script:

```
logrotate -d /etc/logrotate.d/sensu.conf
```

Next steps

Sensu also offers logging of observability event data to a separate JSON log file as a commercial feature. Read the Sensu backend reference for more information about event logging.

Monitor Sensu with Sensu

This guide describes best practices and strategies for monitoring the Sensu backend with another Sensu backend or cluster.

To completely monitor Sensu (a Sensu backend with internal etcd and an agent), you will need at least one independent Sensu instance in addition to the primary instance you want to monitor. The second Sensu instance will ensure that you are notified when the primary is down and vice versa.

This guide requires Sensu plugins using dynamic runtime assets. For more information about using Sensu plugins, read Use dynamic runtime assets to install plugins.

> ***NOTE**: Although this guide describes approaches for monitoring a single backend, these strategies are also useful for monitoring individual members of a backend cluster.*
>
> *This guide does not describe Sensu agent keepalive monitoring.*

The checks in this guide monitor the following ports and endpoints:

Port	Endpoint	Description
2379	`/health`	Etcd health endpoint. Provides health status for etcd nodes.
8080	`/health`	Sensu Go health endpoint. Provides health status for Sensu backends, as well as for PostgreSQL (when enabled).

Register dynamic runtime asset

To power the checks to monitor your Sensu backend, external etcd, and PostgreSQL instances, add the sensu/http-checks dynamic runtime asset. This asset includes the `http-json` plugin, which your checks will rely on.

To register the sensu/http-checks dynamic runtime asset, run:

```
sensuctl asset add sensu/http-checks:0.5.0 -r http-checks
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/http-checks:0.5.0
added asset: sensu/http-checks:0.5.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["http-checks"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: http-checks.

To confirm that the asset is ready to use, run:

```
sensuctl asset list
```

The response should list the renamed http-checks dynamic runtime asset:

```
   Name                  URL                                                                                                  Hash
_____________ ____________________________________________________________________________________________________ _____________

 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_armv7.tar.gz     b28f8c3
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_arm64.tar.gz     7308f9c
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_386.tar.gz       6457583
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_windows_amd64.tar.gz   b936ca0
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_darwin_amd64.tar.gz    38e6cb8
 http-checks   //assets.bonsai.sensu.io/.../http-checks_0.5.0_linux_amd64.tar.gz     bc5fc3b
```

Because plugins are published for multiple platforms, including Linux and Windows, the output will include multiple entries for each of the dynamic runtime assets.

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read the asset reference for more information about dynamic runtime asset builds.*

Monitor your Sensu backend instances

Monitor the host running the `sensu-backend` *locally* by a `sensu-agent` process for operating system checks and metrics.

For Sensu components that must be running for Sensu to create events, you should also monitor the `sensu-backend` remotely from an independent Sensu instance. This will allow you to monitor whether your Sensu event pipeline is working.

To do this, add checks that use the `http-json` plugin from the sensu/http-checks dynamic runtime asset to query Sensu's /health API for your primary (Backend Alpha) and secondary (Backend Beta) backends.

***NOTE**: These examples use the sensu/http-checks dynamic runtime asset. Follow the instructions above to register the sensu/http-checks dynamic runtime asset if you did not previously register it.*

YML

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_beta_backend_health
spec:
  command: http-json --url http://sensu-backend-beta:8080/health --query
".ClusterHealth.[0].Healthy" --expression "== true"
  subscriptions:
    - backend_alpha
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
    - http-checks
```

```
EOF
```

YML

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_alpha_backend_health
spec:
  command: http-json --url http://sensu-backend-alpha:8080/health --query
".ClusterHealth.[0].Healthy" --expression "== true"
  subscriptions:
    - backend_beta
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
    - http-checks
EOF
```

JSON

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_beta_backend_health"
  },
  "spec": {
    "command": "http-json --url http://sensu-backend-beta:8080/health --query
\".ClusterHealth.[0].Healthy\" --expression \"== true\"",
    "subscriptions": [
      "backend_alpha"
    ],
    "interval": 10,
    "publish": true,
    "timeout": 10,
    "runtime_assets": [
```

```
      "http-checks"
    ]
  }
}
EOF
```

JSON

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_alpha_backend_health"
  },
  "spec": {
    "command": "http-json --url http://sensu-backend-alpha:8080/health --query
\".ClusterHealth.[0].Healthy\" --expression \"== true\"",
    "subscriptions": [
      "backend_beta"
    ],
    "interval": 10,
    "publish": true,
    "timeout": 10,
    "runtime_assets": [
      "http-checks"
    ]
  }
}
EOF
```

A successful health check result will be similar to this example:

```
http-json OK:  The value true found at .ClusterHealth.[0].Healthy matched with
expression "== true" and returned true
```

To receive alerts based on your backend health checks, configure a pipeline with event filters and a handler and update your check definitions to reference the pipeline in the pipelines attribute.

Monitor external etcd

If your Sensu Go deployment uses an external etcd cluster, you'll need to check the health of the respective etcd instances for your primary (Backend Alpha) and secondary (Backend Beta) backends.

> ***NOTE**: These examples use the sensu/http-checks dynamic runtime asset. Follow the instructions above to register the sensu/http-checks dynamic runtime asset if you did not previously register it.*

YML

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_beta_etcd_health
spec:
  command: http-json --url http://sensu-etcd-beta:2379/health --query
".ClusterHealth.[0].Healthy" --expression "== true"
  subscriptions:
  - backend_alpha
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
  - http-checks
EOF
```

YML

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_alpha_etcd_health
spec:
  command: http-json --url http://sensu-etcd-alpha:2379/health --query
".ClusterHealth.[0].Healthy" --expression "== true"
```

```
  subscriptions:
  - backend_beta
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
  - http-checks
EOF
```

JSON

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_beta_etcd_health"
  },
  "spec": {
    "command": "http-json --url http://sensu-etcd-beta:2379/health --query
\".ClusterHealth.[0].Healthy\" --expression \"== true\"",
    "subscriptions": [
      "backend_alpha"
    ],
    "interval": 10,
    "publish": true,
    "timeout": 10,
    "runtime_assets": [
      "http-checks"
    ]
  }
}
EOF
```

JSON

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "check_alpha_etcd_health"
  },
  "spec": {
    "command": "http-json --url http://sensu-etcd-alpha:2379/health --query
\".ClusterHealth.[0].Healthy\" --expression \"== true\"",
    "subscriptions": [
      "backend_beta"
    ],
    "interval": 10,
    "publish": true,
    "timeout": 10,
    "runtime_assets": [
      "http-checks"
    ]
  }
}
EOF
```

A successful health check result will be similar to this example:

```
http-json OK:  The value true found at .ClusterHealth.[0].Healthy matched with
expression "== true" and returned true
```

To receive alerts based on your external etcd health checks, configure a pipeline with event filters and a handler and update your check definitions to reference the pipeline in the pipelines attribute.

Monitor PostgreSQL

***COMMERCIAL FEATURE**: Access enterprise-scale PostgreSQL event storage in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

Larger Sensu deployments may use PostgreSQL as an alternative datastore to process larger numbers of events.If you're using PostgreSQL for event storage, we recommend monitoring your PostgreSQL event store's health.

The connection to PostgreSQL is exposed on Sensu's `/health` endpoint, which provides information

about the event store's health. PostgreSQL data from the `/health` endpoint will look like this example:


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 3470366781180380700,
      "MemberIDHex": "302938336092857e",
      "Name": "sensu00",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 15883454222313069000,
      "MemberIDHex": "dc6d5d7607261af7",
      "Name": "sensu01",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 11377294497886210000,
      "MemberIDHex": "9de44510fb838bbd",
      "Name": "sensu02",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 13239446193995635000,
    "member_id": 3470366781180380700,
    "raft_term": 1549
  },
  "PostgresHealth": [
    {
      "Name": "sensu_postgres",
      "Active": true,
      "Healthy": true
    }
  ]
}
```

To monitor PostgreSQL's health from Sensu, create checks that use the `http-json` plugin from the sensu/http-checks dynamic runtime asset.

> ***NOTE**: These examples use the sensu/http-checks dynamic runtime asset. Follow the instructions above to register the sensu/http-checks dynamic runtime asset if you did not previously register it.*

After you register the sensu/http-checks dynamic runtime asset, create two checks ("healthy" and "active") to monitor PostgreSQL's health from Sensu. Make sure to update the `--url` value with your backend address before running the commands to create the checks.

Run the following command to add the "healthy" check:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: postgres_healthy_http_check
spec:
  command: http-json --url https://sensu.example.com:8080/health --query
".PostgresHealth.[0].Healthy" --expression "== true"
  round_robin: true
  publish: true
  interval: 60
  subscriptions:
  - system
  runtime_assets:
  - http-checks
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "postgres_healthy_http_check"
```

```
  },
  "spec": {
    "command": "http-json --url https://sensu.example.com:8080/health --query
\".PostgresHealth.[0].Healthy\" --expression \"== true\"",
    "round_robin": true,
    "publish": true,
    "interval": 60,
    "subscriptions": [
      "system"
    ],
    "runtime_assets": [
      "http-checks"
    ]
  }
}
EOF
```

Run the following command to add the “active” check:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: postgres_active_http_check
spec:
  command: http-json --url https://sensu.example.com:8080/health --query
".PostgresHealth.[0].Active" --expression "== true"
  round_robin: true
  publish: true
  interval: 60
  subscriptions:
  - system
  runtime_assets:
  - http-checks
EOF
```

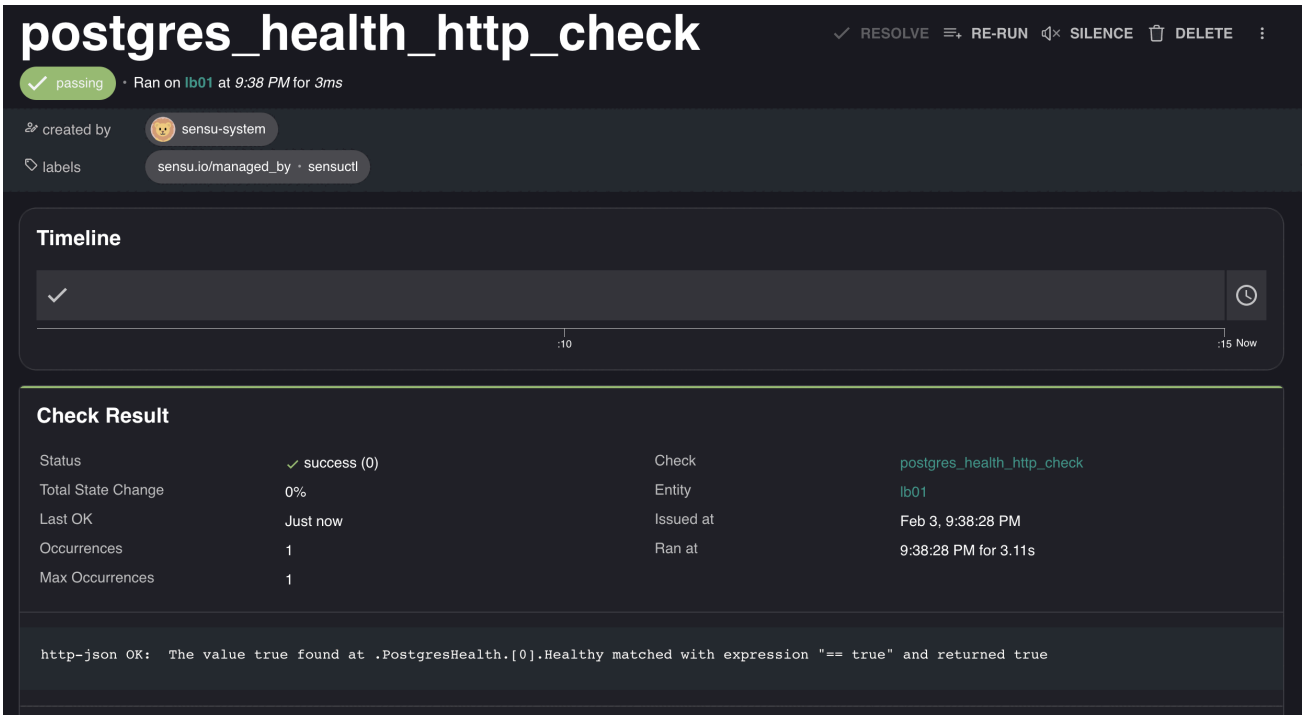
SHELL

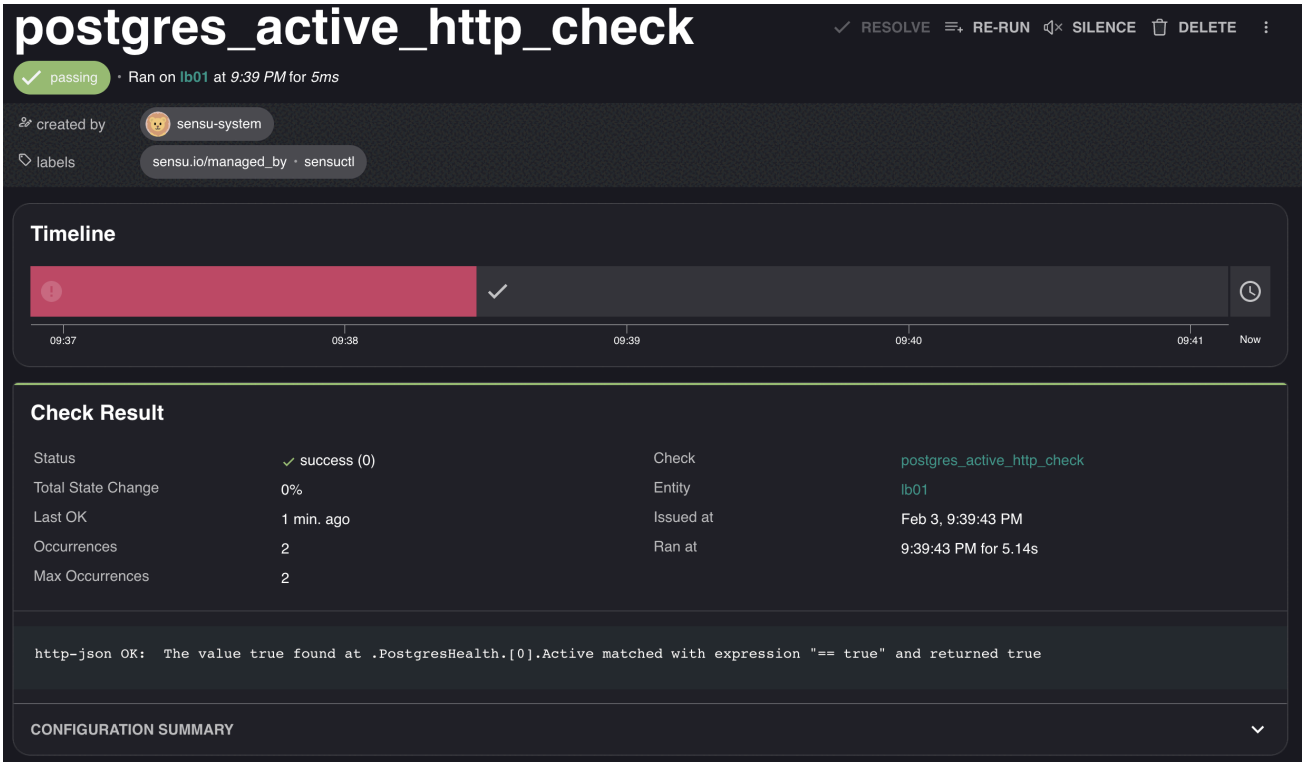
```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "postgres_active_http_check"
  },
  "spec": {
    "command": "http-json --url https://sensu.example.com:8080/health --query
\".PostgresHealth.[0].Active\" --expression \"== true\"",
    "round_robin": true,
    "publish": true,
    "interval": 60,
    "subscriptions": [
      "system"
    ],
    "runtime_assets": [
      "http-checks"
    ]
  }
}
EOF
```

Successful PostgreSQL health check results will be similar to this example:

```
http-json OK:  The value true found at .PostgresHealth.[0].Healthy matched with
expression "== true" and returned true
http-json OK:  The value true found at .PostgresHealth.[0].Active matched with
expression "== true" and returned true
```

In the Sensu web UI, you should see check results similar to these examples:





To receive alerts based on your PostgreSQL health check, configure a pipeline with event filters and a handler and update your check definition to reference the pipeline in the pipelines attribute.

Health reference

Use Sensu's /health API to make sure your backend is up and running and check the health of your etcd cluster members and PostgreSQL datastore resources.

A request to the /health API endpoint retrieves a JSON map with health data for your Sensu instance. Here's an example request to the health endpoint:

```
curl -X GET \
http://127.0.0.1:8080/health
```

Healthy cluster example

In this example, all cluster members are healthy.


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 9861478486968594000,
      "MemberIDHex": "88db026f7feb72b4",
      "Name": "backend01",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 16828500076473182000,
      "MemberIDHex": "e98ad7a888d16bd6",
      "Name": "backend02",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 848052855499371400,
```

```
      "MemberIDHex": "bc4e39432cbb36d",
      "Name": "backend03",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 17701109828877156000,
    "member_id": 16828500076473182000,
    "raft_term": 42
  }
},
  "PostgresHealth": [
    {
      "Name": "my-first-postgres",
      "Active": true,
      "Healthy": true
    },
    {
      "Name": "my-other-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```


Unhealthy cluster member example

In this example, one cluster member is unhealthy: it cannot communicate with the other cluster members.


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 9861478486968594000,
      "MemberIDHex": "88db026f7feb72b4",
      "Name": "backend01",
      "Err": "context deadline exceeded",
```

```
      "Healthy": false
    },
    {
      "MemberID": 16828500076473182000,
      "MemberIDHex": "e98ad7a888d16bd6",
      "Name": "backend02",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 848052855499371400,
      "MemberIDHex": "bc4e39432cbb36d",
      "Name": "backend03",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 17701109828877156000,
    "member_id": 16828500076473182000,
    "raft_term": 42
  }
},
  "PostgresHealth": [
    {
      "Name": "my-first-postgres",
      "Active": true,
      "Healthy": true
    },
    {
      "Name": "my-other-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```


***NOTE**: The HTTP response codes for the health endpoint indicate whether your request reached Sensu rather than the health of your Sensu instance. In this example, even though the cluster is unhealthy, the request itself reached Sensu, so the response code is `200 OK`. To determine the health of your Sensu instance, you must process the JSON response body. The health specification*

describes each attribute in the response body.

Health specification

Top-level attributes

Alarms	
description	Top-level attribute that lists all active etcd alarms.
required	true
type	String
example	`"Alarms": null`

ClusterHealth	
description	Top-level attribute that includes health status information for every etcd cluster member.
required	true
type	Map of key-value pairs
example	<pre>"ClusterHealth": [{ "MemberID": 2882886652148554927, "MemberIDHex": "8923110df66458af", "Name": "default", "Err": "", "Healthy": true }]</pre>

Header

description	Top-level map that includes the response header for the entire cluster response.
required	true
type	Map of key-value pairs
example	`"Header": { "cluster_id": 4255616344056076734, "member_id": 2882886652148554927, "raft_term": 26 }`

PostgresHealth

description	Top-level map that includes health information for PostgreSQL resources. If your Sensu instance is not configured to use a PostgreSQL datastore, the health payload will not include `PostgresHealth`.
type	Map of key-value pairs
example	`"PostgresHealth": [ { "Name": "postgres-test", "Active": false, "Healthy": false }, { "Name": "postgres", "Active": true, "Healthy": true } ]`

ClusterHealth attributes

Err	
description	Any errors Sensu encountered while checking the etcd cluster member's health.
required	true
type	String
example	`"Err": ""`

Healthy	
description	`true` if the etcd cluster member is connected. Otherwise, `false`.
required	true
type	Boolean
default	`false`
example	`"Healthy": true`

MemberID	
description	The etcd cluster member's ID.
required	true
type	Integer
example	

```
"MemberID": 2882886652148554927
```

MemberIDHex	
description	The hexadecimal representation of the etcd cluster member's ID.
required	true
type	String
example	`"MemberIDHex": "8923110df66458af"`

Name	
description	The etcd cluster member's name.
required	true
type	String
example	`"Name": "default"`

Header attributes

cluster_id	
description	The etcd cluster ID.
required	true
type	Integer
example	`"cluster_id": 4255616344056076734`

member_id	
description	The etcd cluster member's ID.
required	true
type	Integer
example	`"member_id": 2882886652148554927`

raft_term	
description	The etcd cluster member's raft term.
required	true
type	Integer
example	`"raft_term": 26`

PostgresHealth attributes

Active	
description	`true` if the datastore is configured to use the PostgreSQL configuration. Otherwise, `false`.
required	true
type	Boolean
default	`false`

example	<pre>"Active": true</pre>

Healthy

description	`true` if the PostgreSQL datastore is connected and can query the events table. Otherwise, `false`.
required	true
type	Boolean
default	`false`
example	<pre>"Healthy": true</pre>

Name

description	The PostgreSQL configuration resource. Sensu retrieves the `Name` from datastore metadata.
required	true
type	String
example	<pre>"Name": "postgres"</pre>

Ready reference

Use Sensu's /ready API endpoint to confirm whether a Sensu instance is ready to serve API requests and accept agent connections.

A request to the /ready backend API endpoint retrieves a text response with information about whether your Sensu instance is ready to serve API requests.Ħere's an example request to the /ready API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/ready
```

A request to the /ready agent transport API endpoint via the backend WebSocket retrieves information about whether your Sensu instance is ready to accept agent connections.Ħere's an example request to the /ready agent transport API endpoint using the default WebSocket port 8081:

```
curl -X GET \
http://127.0.0.1:8081/ready
```

Ready response example

The following response means that the Sensu instance is ready to serve API requests or accept agent connections:

```
ready
```

Not ready response examples

To help prevent instability during sensu-backend startup, use the `api-serve-wait-time` and

`agent-serve-wait-time` backend configuration options.

Use `api-serve-wait-time` to configure a delay after startup before the backend API will serve traffic. Until the specified duration expires, the text response body will state that the API is unavailable:

```
API unavailable during startup.
See api-serve-wait-time settings.
```

Use `agent-serve-wait-time` to configure a delay after startup before the agent listener will begin accepting agent connections.Until the specified duration expires, the text response body will state that agentd is unavailable:

```
agentd temporarily unavailable during startup
```

Not-ready responses include a `Retry-After` header that lists the specified `api-serve-wait-time` or `agent-serve-wait-time` duration.

Tessen reference

Tessen is the Sensu call-home service. It is enabled by default on Sensu backends. Tessen sends anonymized data about Sensu instances to Sensu Inc., including the version, cluster size, number of events processed, and number of resources created (like checks and handlers). We rely on Tessen data to understand how Sensu is being used and make informed decisions about product improvements. Read Announcing Tessen, the Sensu call-home service to learn more about Tessen.

All data submissions are logged for complete transparency at the `info` log level and transmitted over HTTPS. Read Troubleshoot Sensu to set the Sensu backend log level and view logs.

Configure Tessen

You can use core/v2/tessen and sensuctl to view your Tessen configuration. If you are using an unlicensed Sensu instances, you can also use core/v2/tessen and sensuctl to opt in or opt out of Tessen.

> ***NOTE**: Tessen is enabled by default on Sensu backends and required for licensed Sensu instances. If you have a licensed instance and want to opt out of Tessen, contact your account manager.*

To manage Tessen configuration for your unlicensed instance with sensuctl, configure sensuctl as the default `admin` user.

To view Tessen status:

```
sensuctl tessen info
```

To opt out of Tessen:

```
sensuctl tessen opt-out
```

***NOTE**: For licensed Sensu instances, the Tessen configuration setting will automatically override to `opt-in` at runtime.*

You can use the `--skip-confirm` flag to skip the confirmation step:

```
sensuctl tessen opt-out --skip-confirm
```

To opt in to Tessen:

```
sensuctl tessen opt-in
```

Tessen specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For Tessen configuration in this version of Sensu, the `api_version` should always be `core/v2`.
required	Required for Tessen configuration in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: core/v2` **JSON** `{ "api_version": "core/v2" }`

spec

description	Top-level map that includes Tessen configuration spec attributes.
required	Required for Tessen configuration in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** ```yaml spec: opt_out: false ``` **JSON** ```json { "spec": { "opt_out": false } } ```

type

description	Top-level attribute that specifies the `sensuctl create` resource type. Tessen configuration should always be type `TessenConfig`.
required	Required for Tessen configuration in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** ```yaml type: TessenConfig ```

JSON

```
{
  "type": "TessenConfig"
}
```

Spec attributes

opt_out	
description	`true` to opt out of Tessen. Otherwise, `false`. Tessen is enabled by default on Sensu backends and required for licensed Sensu instances.
required	true
type	Boolean
default	`false`
example	**YML** `opt_out: false` **JSON** `{ "opt_out": false }`

Tessen configuration example

This example is in `wrapped-json` format for use with `sensuctl create`. To manage Tessen for unlicensed Sensu instances with the Tessen API, use non-wrapped `json` format as shown in the API docs.

YML

```
---
type: TessenConfig
api_version: core/v2
spec:
  opt_out: false
```

JSON

```
{
  "type": "TessenConfig",
  "api_version": "core/v2",
  "spec": {
    "opt_out": false
  }
}
```

Tessen metrics log examples

For unlicensed instances that opt in to Tessen and all licensed instances, Sensu sends various metrics back to the Tessen service. In the example metrics log below, Sensu is sending the number of check hooks back to the Tessen service.

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "hook_count",
    "metric_value": 2,
    "msg": "collected a metric for tessen",
    "time": "2019-09-16T09:02:11Z"
}
```

Sensu also sends other metrics, such as the number of handlers:

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "handler_count",
    "metric_value": 10,
    "msg": "collected a metric for tessen",
    "time": "2019-09-16T09:02:06Z"
}
```


Or the number of filters:


```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "filter_count",
    "metric_value": 4,
    "msg": "collected a metric for tessen",
    "time": "2019-09-16T09:02:01Z"
}
```


Or the number of authentication providers, secrets providers, and secrets:


```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "auth_provider_count",
    "metric_value": 2,
    "msg": "collected a metric for tessen",
    "time": "2020-03-30T15:16:42-04:00"
}
```

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "secret_provider_count",
    "metric_value": 1,
    "msg": "collected a metric for tessen",
```

```
    "time": "2020-03-30T15:17:12-04:00"
}
```

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "secret_count",
    "metric_value": 1,
    "msg": "collected a metric for tessen",
    "time": "2020-03-30T15:16:17-04:00"
}
```

If you opt into Tessen, you can view all of the metrics in the logs:

```
journalctl _COMM=sensu-backend.service
```

To view the events on-disk, read Log Sensu services with systemd.

Manage Secrets

Sensu's secrets management eliminates the need to expose secrets like usernames, passwords, and access keys in your Sensu configuration. Secrets management is available for Sensu handler, mutator, and check resources.

Use secrets management in Sensu explains how to use Sensu's secrets provider (`Env`) or HashiCorp Vault as your external secrets provider and authenticate without exposing your secrets. Follow this guide to set up your PagerDuty Integration Key as a secret and create a PagerDuty handler definition that requires the secret. Your Sensu backend will be able to execute the handler with any check.

Secrets

Secrets are configured with Sensu's `Secret` resources. A secret resource definition refers to the secrets provider and an ID (the named secret to fetch from the secrets provider).

The secrets reference includes the specification, sensuctl configuration subcommands, and examples for secrets resources.

Secrets providers

The Sensu Go commercial distribution includes a secrets provider, `Env` , that exposes secrets from environment variables on your Sensu backend nodes. You can also use the secrets provider `VaultProvider` to authenticate via the HashiCorp Vault integration.

The secrets providers reference includes the resource specification, instructions for retrieving your secrets providers configuration via the Sensu API, and examples.

Use secrets management in Sensu

***COMMERCIAL FEATURE**: Access the Env and VaultProvider secrets provider datatypes in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu's secrets management allows you to avoid exposing secrets like usernames, passwords, and access keys in your Sensu configuration. In this guide, you'll learn how to use Sensu's `Env` secrets provider or HashiCorp Vault as your external secrets provider and authenticate without exposing your secrets. You'll set up your PagerDuty Integration Key as a secret, create a PagerDuty handler definition that requires the secret, and configure a pipeline that includes the PagerDuty handler. Your Sensu backend can then execute the pipeline with any check.

To follow this guide, you'll need to install the Sensu backend, have at least one Sensu agent running, and install and configure sensuctl.

Secrets are configured via secrets resources. A secret resource definition refers to the secrets provider (`Env` or `VaultProvider`) and an ID (the named secret to fetch from the secrets provider).

This guide only covers the handler use case, but you can use secrets management in handler, mutator, and check execution. When a check configuration references a secret, the Sensu backend will only transmit the check's execution requests to agents that are connected via mutually authenticated transport layer security (mTLS)-encrypted WebSockets.

The secret included in your Sensu handler will be exposed to Sensu services at runtime as an environment variable. Sensu only exposes secrets to Sensu services like environment variables and automatically redacts secrets from all logs, the API, and the web UI.

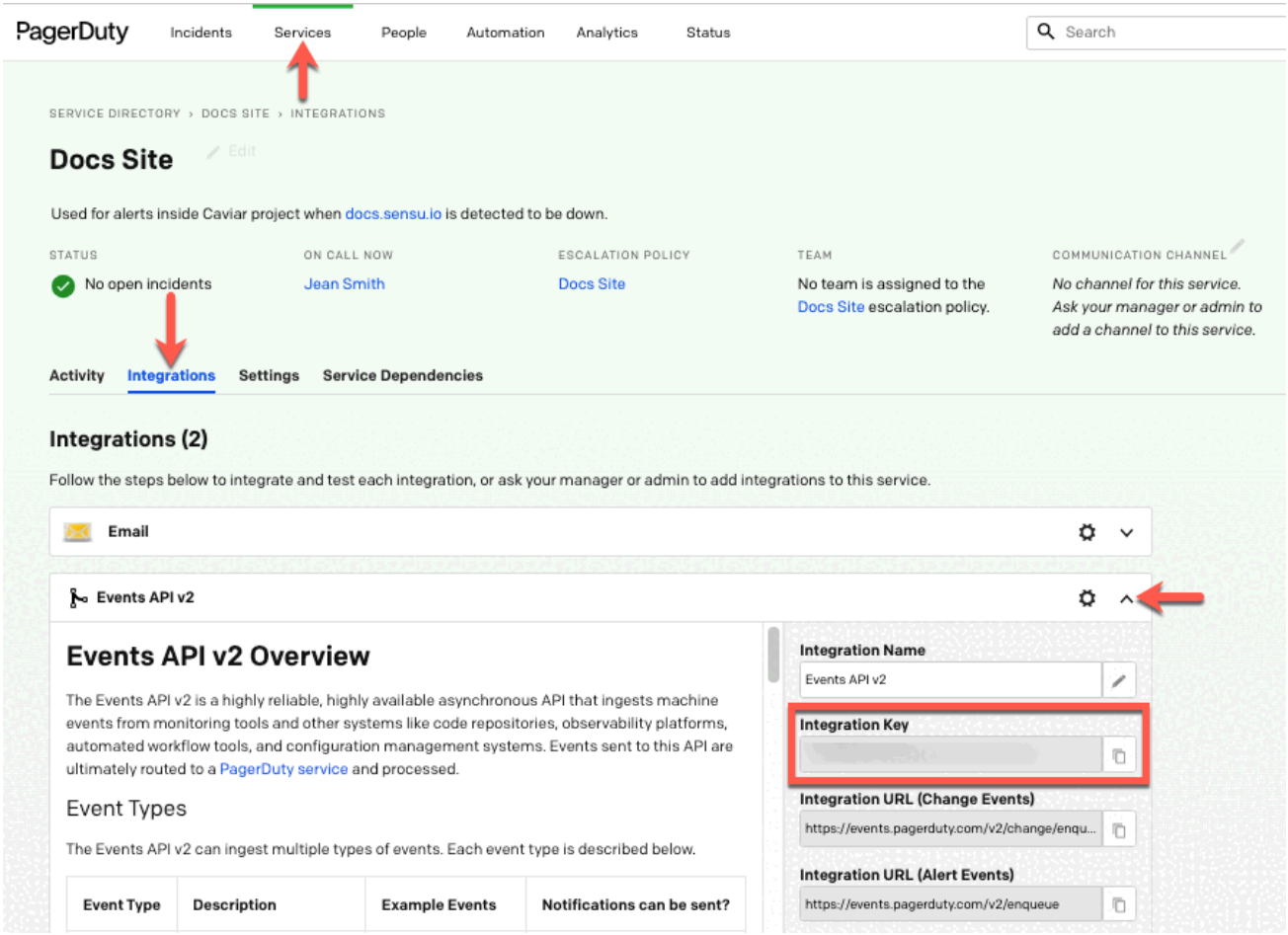
Retrieve your PagerDuty Integration Key

The example in this guide uses the PagerDuty Integration Key as a secret and a PagerDuty handler definition that requires the secret.

Here's how to find your Integration Key in PagerDuty so you can set it up as your secret:

1. Log in to your PagerDuty account.
2. In the **Services** drop-down menu, select **Service Directory**.

3. Enter the name of your Sensu service in the search field.
4. Click to select your Sensu service from the list of search results.
5. Click the **Integrations** tab.
6. Click the drop-down arrow for the **Events API**. The Integration Key is listed in the second field.



Make a note of your Integration Key — you'll need it to create your backend environment variable or HashiCorp Vault secret.

Use Env for secrets management

The Sensu Go commercial distribution includes a secrets provider, `Env`, that exposes secrets from environment variables on your Sensu backend nodes. The `Env` secrets provider is automatically created with an empty `spec` when you start your Sensu backend.

Create your backend environment variable

To use the `Env` secrets provider, add your secret as a backend environment variable.

First, make sure you have created the files you need to store backend environment variables.

Then, run the following code, replacing `INTEGRATION_KEY` with your PagerDuty Integration Key:

SHELL

```
echo 'SENSU_PAGERDUTY_KEY=INTEGRATION_KEY' | sudo tee -a /etc/default/sensu-backend
```

SHELL

```
echo 'SENSU_PAGERDUTY_KEY=INTEGRATION_KEY' | sudo tee -a /etc/sysconfig/sensu-backend
```

Restart the sensu-backend:

```
sudo systemctl restart sensu-backend
```

This configures the `SENSU_PAGERDUTY_KEY` environment variable to your PagerDuty Integration Key in the context of the sensu-backend process.

Create your Env secret

Now you'll use `sensuctl create` to create your secret. This code creates a secret named `pagerduty_key` that refers to the environment variable ID `SENSU_PAGERDUTY_KEY`. Run:

SHELL

```
cat << EOF | sensuctl create
---
type: Secret
api_version: secrets/v1
metadata:
  name: pagerduty_key
spec:
  id: SENSU_PAGERDUTY_KEY
  provider: env
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "pagerduty_key"
  },
  "spec": {
    "id": "SENSU_PAGERDUTY_KEY",
    "provider": "env"
  }
}
EOF
```

You can securely pass your PagerDuty Integration Key in Sensu checks, handlers, and mutators by referring to the `pagerduty_key` secret. Skip to the add a handler section, where you'll use your `pagerduty_key` secret in your handler definition.

Use HashiCorp Vault for secrets management

This section explains how to use HashiCorp Vault as your external secrets provider to authenticate via the HashiCorp Vault integration's token auth method or TLS certificate auth method.

> ***NOTE**: You must set up HashiCorp Vault to use `VaultProvider` secrets management in production. The examples in this guide use the Vault dev server, which is useful for learning and experimenting. The Vault dev server gives you access to a preconfigured, running Vault server with in-memory storage that you can use right away. Follow the HashiCorp Learn curriculum when you are ready to set up a production server in Vault.*
>
> *In addition, this guide uses the Vault KV secrets engine. Using the Vault KV secrets engine with the Vault dev server requires v2 connections. For this reason, in the `VaultProvider` spec in these examples, the client `version` value is **v2**.*

Configure your Vault authentication method (token or TLS)

If you use HashiCorp Vault as your external secrets provider, you can authenticate via the HashiCorp

Vault integration's token or transport layer security (TLS) certificate authentication method.

Vault token authentication

Follow the steps in this section to use HashiCorp Vault as your external secrets provider to authenticate with the HashiCorp Vault integration's token auth method.

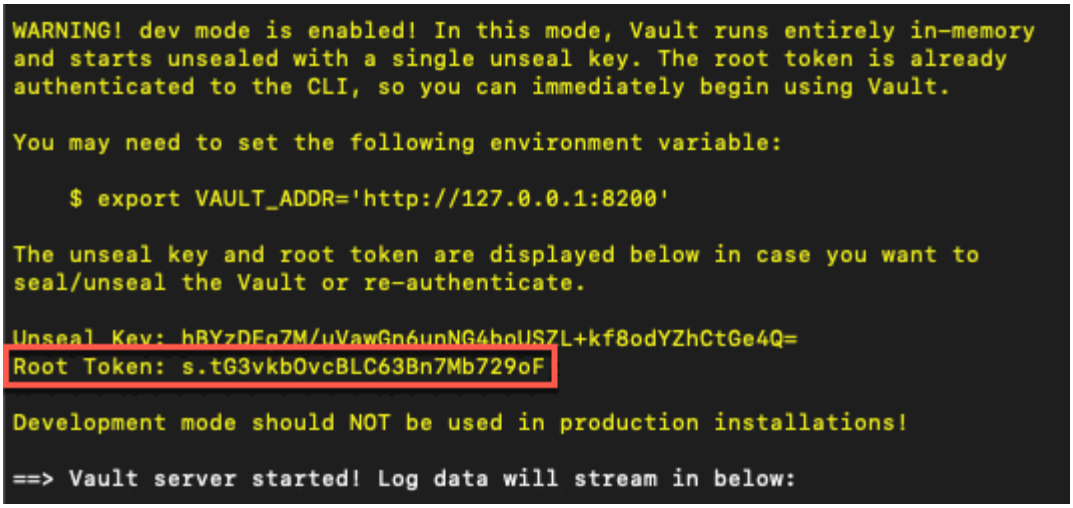
Retrieve your Vault root token

> ***NOTE**: The examples in this guide use the `Root Token` for the the Vault dev server, which gives you access to a preconfigured, running Vault server with in-memory storage that you can use right away. Follow the HashiCorp Learn curriculum when you are ready to set up a production server in Vault.*

To retrieve your Vault root token:

1. Download and install the Vault edition for your operating system.
2. Open a terminal window and run `vault server -dev`.

The command output includes a `Root Token` line. Find this line in your command output and copy the `Root Token` value. You will use it next to create your Vault secrets provider.



Leave the Vault dev server running. Because you aren't using TLS, you will need to set `VAULT_ADDR=http://127.0.0.1:8200` in your shell environment.

Create your Vault secrets provider

***NOTE**: In Vault's dev server, TLS is not enabled, so you won't be able to use certificate-based authentication.*

Use `sensuctl create` to create your secrets provider, `vault`. In the code below, replace `<root_token>` with the `Root Token` value for your Vault dev server. Then, run:

SHELL

```
cat << EOF | sensuctl create
---
type: VaultProvider
api_version: secrets/v1
metadata:
  name: vault
spec:
  client:
    address: http://localhost:8200
    token: <root_token>
    version: v2
    tls: null
    max_retries: 2
    timeout: 20s
    rate_limiter:
      limit: 10
      burst: 100
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "vault"
  },
  "spec": {
    "client": {
      "address": "http://localhost:8200",
      "token": "<root_token>",
      "version": "v2",
```

```
      "tls": null,
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10,
        "burst": 100
      }
    }
  }
}
EOF
```

To continue, skip ahead to create your Vault secret.

Vault TLS certificate authentication

This section explains how use HashiCorp Vault as your external secrets provider to authenticate with the HashiCorp Vault integration's TLS certificate auth method.

> ***NOTE**: You will need to set up HashiCorp Vault in production to use TLS certificate-based authentication. In Vault's dev server, TLS is not enabled. Follow the HashiCorp Learn curriculum when you are ready to set up a production server in Vault.*

First, in your Vault, enable and configure certificate authentication. For example, your Vault might be configured for certificate authentication like this:

```
vault write auth/cert/certs/sensu-backend \
    display_name=sensu-backend \
    policies=sensu-backend-policy \
    certificate=@sensu-backend-vault.pem \
    ttl=3600
```

Second, configure your `VaultProvider` in Sensu:

YML

```
---
```

```
type: VaultProvider
api_version: secrets/v1
metadata:
  name: vault
spec:
  client:
    address: https://vault.example.com:8200
    version: v2
    tls:
      ca_cert: /path/to/your/ca.pem
      client_cert: /etc/sensu/ssl/sensu-backend-vault.pem
      client_key: /etc/sensu/ssl/sensu-backend-vault-key.pem
      cname: sensu-backend.example.com
    max_retries: 2
    timeout: 20s
    rate_limiter:
      limit: 10
      burst: 100
```

SHELL

```
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "vault"
  },
  "spec": {
    "client": {
      "address": "https://vault.example.com:8200",
      "version": "v2",
      "tls": {
        "ca_cert": "/path/to/your/ca.pem",
        "client_cert": "/etc/sensu/ssl/sensu-backend-vault.pem",
        "client_key": "/etc/sensu/ssl/sensu-backend-vault-key.pem",
        "cname": "sensu-backend.example.com"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10,
        "burst": 100
```

```
        }
      }
    }
}
```

The certificate you specify for `tls.client_cert` should be the same certificate you configured in your Vault for certificate authentication.

Next, create your Vault secret.

Create your Vault secret

First, retrieve your PagerDuty Integration Key (the secret you will set up in Vault).

Next, open a new terminal and run `vault kv put secret/pagerduty key=<integration_key>`. Replace `<integration_key>` with your PagerDuty Integration Key. This writes your secret into Vault.

In this example, the name of the secret is `pagerduty`. The `pagerduty` secret contains a key, and you specified that the `key` value is your PagerDuty Integration Key.

> ***NOTE**: The Vault dev server is preconfigured with the secret keyspace already set up, so we recommend using the `secret/` path for the `id` value while you are learning and getting started with Vault secrets management.*
>
> *This example uses the `id` format for the Vault KV Secrets Engine v1: `secret/pagerduty#key`. If you are using the Vault KV Secrets Engine v2, the format is `secrets/sensu#pagerduty#key`.*

Run `vault kv get secret/pagerduty` to view the secret you just set up.

Use `sensuctl create` to create your `vault` secret:

SHELL

```
cat << EOF | sensuctl create
---
type: Secret
api_version: secrets/v1
metadata:
```

```
  name: pagerduty_key
spec:
  id: secret/pagerduty#key
  provider: vault
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "pagerduty_key"
  },
  "spec": {
    "id": "secret/pagerduty#key",
    "provider": "vault"
  }
}
EOF
```

Now you can securely pass your PagerDuty Integration Key in the handlers, and mutators by referring to the `pagerduty_key` secret.In the add a handler section, you'll use your `pagerduty_key` secret in your handler definition.

Add a handler

Register the sensu/sensu-pagerduty-handler dynamic runtime asset

To begin, register the sensu/sensu-pagerduty-handler dynamic runtime asset with `sensuctl asset add` :

```
sensuctl asset add sensu/sensu-pagerduty-handler:2.2.0 -r pagerduty-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset: `pagerduty-handler`.

> ***NOTE**: You can adjust the dynamic runtime asset definition according to your Sensu configuration if needed.*

Run `sensuctl asset list --format yaml` to confirm that the dynamic runtime asset is ready to use.

With this handler, Sensu can trigger and resolve PagerDuty incidents.However, you still need to add your secret to the handler spec so that it requires your backend to request secrets from your secrets provider.

Add your secret to the handler spec

To create a handler definition that uses your `pagerduty_key` secret, run:

SHELL

```
cat << EOF | sensuctl create
---
api_version: core/v2
type: Handler
metadata:
  name: pagerduty
spec:
  type: pipe
  command: pagerduty-handler --token $PD_TOKEN
  secrets:
  - name: PD_TOKEN
    secret: pagerduty_key
  runtime_assets:
  - pagerduty-handler
  timeout: 10
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
```

```
  "api_version": "core/v2",
  "type": "Handler",
  "metadata": {
    "name": "pagerduty"
  },
  "spec": {
    "type": "pipe",
    "command": "pagerduty-handler --token $PD_TOKEN",
    "secrets": [
      {
        "name": "PD_TOKEN",
        "secret": "pagerduty_key"
      }
    ],
    "runtime_assets": [
      "pagerduty-handler"
    ],
    "timeout": 10
  }
}
EOF
```

Configure a pipeline

Now that your handler is set up and Sensu can create incidents in PagerDuty, you can configure a pipeline to start receiving alerts based on the events your checks create. A single pipeline workflow can include one or more filters, one mutator, and one handler.

In this case, the pipeline will include the built-in is_incident event filter and the `pagerduty` handler you created in the previous step. You can add this pipeline to any check to receive a PagerDuty alert for every warning (`1`) or critical (`2`) event the check generates, as well as for resolution events.

To create the pipeline, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Pipeline
api_version: core/v2
```

```
metadata:
  name: incident_alerts
spec:
  workflows:
  - name: pagerduty_incidents
    filters:
    - name: is_incident
      type: EventFilter
      api_version: core/v2
    handler:
      name: pagerduty
      type: Handler
      api_version: core/v2
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Pipeline",
  "api_version": "core/v2",
  "metadata": {
    "name": "incident_alerts"
  },
  "spec": {
    "workflows": [
      {
        "name": "pagerduty_incidents",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "handler": {
          "name": "pagerduty",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
```

```
  }
}
EOF
```

To automate this workflow, include the `incident_alerts` pipeline in any Sensu check definition in the check's pipelines attribute. When you list a pipeline in a check definition, all the observability events that the check produces will be processed according to the pipeline's workflows.

Next steps

Add your pipeline to any check to start receiving PagerDuty alerts based on observability event data. Read Send PagerDuty alerts with Sensu for an example that shows how to edit a check definition to add a pipeline.

Read the secrets or secrets providers reference for in-depth secrets management documentation.

Secrets reference

***COMMERCIAL FEATURE**: Access the Secret datatype in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu's secrets management eliminates the need to expose secrets in your Sensu configuration. When a Sensu resource definition requires a secret (for example, a username or password), Sensu allows you to obtain secrets from one or more external secrets providers, so you can both refer to external secrets and consume secrets via backend environment variables.

***NOTE**: Secrets management is implemented for checks, handlers, and mutators.*

Only Sensu backends have access to request secrets from a secrets provider. Sensu backends cache fetched secrets in memory, with no persistence to a Sensu datastore or file on disk. Secrets provided via a "lease" with a "lease duration" are deleted from Sensu's in-memory cache after the configured number of seconds, prompting the Sensu backend to request the secret again.

Secrets are only transmitted over a transport layer security (TLS) WebSocket connection. Unencrypted connections must not transmit privileged information. For checks, hooks, and dynamic runtime assets, you must enable mutual TLS (mTLS). Sensu will not transmit secrets to agents that do not use mTLS.

Sensu only exposes secrets to Sensu services like environment variables and automatically redacts secrets from all logs, the API, and the web UI.

Secret examples

A secret resource definition refers to a secrets `id` and a secrets `provider`. Read the secrets provider reference for the provider specification.

YML

```
---
type: Secret
api_version: secrets/v1
metadata:
```

```
  name: sensu-ansible-token
spec:
  id: ANSIBLE_TOKEN
  provider: env
```

JSON

```
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible-token"
  },
  "spec": {
    "id": "ANSIBLE_TOKEN",
    "provider": "env"
  }
}
```

Configure secrets that target a HashiCorp Vault as shown in the following example:

YML

```
---
type: Secret
api_version: secrets/v1
metadata:
  name: sensu-ansible
spec:
  id: 'secret/database#password'
  provider: vault
```

JSON

```
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible"
  },
```

```
  "spec": {
    "id": "secret/database#password",
    "provider": "vault"
  }
}
```

The `id` value for secrets that target a HashiCorp Vault must start with the name of the secret's path in Vault. The Vault dev server is preconfigured with the secret keyspace already set up. This is convenient for learning and getting started with Vault secrets management, so this example and our guide to secrets management use the `secret/` path for the `id` value. In this example, the name of the secret is `database`. The database secret contains a key called `password,` and its value is the password to our database.

Secret configuration

You can use the enterprise/secrets/v1 API endpoints and sensuctl to create, view, and manage your secrets configuration. To manage secrets configuration with sensuctl, configure sensuctl as the default `admin` user.

The standard sensuctl subcommands are available for secrets (list, info, and delete).

To list all secrets:

```
sensuctl secret list
```

To review a secret's status:

```
sensuctl secret info SECRET_NAME
```

To delete a secret:

```
sensuctl secret delete SECRET_NAME
```

`SECRET_NAME` is the value specified in the secret's `name` metadata attribute.

Secret specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For secrets configuration in this version of Sensu, the api_version should always be `secrets/v1`.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `api_version: secrets/v1` **JSON** `{ "api_version": "secrets/v1" }`

metadata	
description	Top-level scope that contains the secret's `name` and `namespace` as well as the `created_by` field.
required	true
type	Map of key-value pairs
example	**YML** `metadata:`

```
  name: sensu-ansible-token
  namespace: default
  created_by: admin
```

JSON

```
{
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

spec	
description	Top-level map that includes secrets configuration spec attributes.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs
example	**YML** `spec:` `  id: ANSIBLE_TOKEN` `  provider: env`

JSON

```
{
  "spec": {
    "id": "ANSIBLE_TOKEN",
    "provider": "env"
  }
}
```

type	
description	Top-level attribute that specifies the resource type. For secrets configuration, the type should always be `Secret`.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `type: Secret` **JSON** `{ "type": "Secret" }`

Metadata attributes

created_by	
description	Username of the Sensu user who created the secret or last updated the secret. Sensu automatically populates the `created_by` field when the secret is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON**

```
{
  "created_by": "admin"
}
```

name

description	Name for the secret that is used internally by Sensu.
required	true
type	String
example	**YML** `name: sensu-ansible-token` **JSON** `{ "name": "sensu-ansible-token" }`

namespace

description	Sensu RBAC namespace that the secret belongs to.
required	true
type	String
example	**YML** `namespace: default` **JSON** `{`

```
  "namespace": "default"
}
```

Spec attributes

id	
description	Identifying key for the provider to use to retrieve the secret. For the `Env` secrets provider, the `id` is the environment variable. For the `VaultProvider` secrets provider, the `id` specifies the secrets engine path, the path to the secret within that secrets engine, and the field to retrieve within the secret.
required	true
type	String
example for Vault KV Secrets Engine v1	**YML** `id: secret/ansible#token` **JSON** `{ "id": "secret/ansible#token" }`
example for Vault KV Secrets Engine v2	**YML** `id: secrets/sensu#ansible#token` **JSON** `{ "id": "secrets/sensu#ansible#token" }`

provider	
description	Name of the provider with the secret.
required	true
type	String
example	**YML** `provider: vault` **JSON** `{ "provider": "vault" }`

Secrets providers reference

***COMMERCIAL FEATURE**: Access the Env and VaultProvider secrets provider datatypes in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu's secrets management eliminates the need to expose secrets like usernames, passwords, and access keys in your Sensu configuration. With Sensu's secrets management, you can obtain secrets from one or more external secrets providers, refer to external secrets, and consume secrets via backend environment variables.

***NOTE**: Secrets management is implemented for checks, handlers, and mutators.*

Only Sensu backends have access to request secrets from a secrets provider. Secrets are only transmitted over a transport layer security (TLS) WebSocket connection. Unencrypted connections must not transmit privileged information. For checks, hooks, and dynamic runtime assets, you must enable mutual TLS (mTLS). Sensu will not transmit secrets to agents that do not use mTLS.

The Sensu Go commercial distribution includes a secrets provider, `Env`, that exposes secrets from environment variables on your Sensu backend nodes. You can also use the secrets provider `VaultProvider` to authenticate via the HashiCorp Vault integration's token auth method or TLS certificate auth method.

You can configure any number of `VaultProvider` secrets providers. However, you can only have a single `Env` secrets provider: the one that is included with the Sensu Go commercial distribution.

Secrets providers are cluster-wide resources and compatible with generic functions.

VaultProvider secrets provider example

The `VaultProvider` secrets provider is a vendor-specific implementation for HashiCorp Vault secrets management.

YML

```
---
```

```yaml
type: VaultProvider
api_version: secrets/v1
metadata:
  name: vault
spec:
  client:
    address: https://vaultserver.example.com:8200
    token: VAULT_TOKEN
    version: v1
    tls:
      ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
    max_retries: 2
    timeout: 20s
    rate_limiter:
      limit: 10
      burst: 100
```

JSON

```json
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "vault"
  },
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
```

Env secrets provider example

Sensu's `Env` secrets provider exposes secrets from backend environment variables. The `Env` secret provider is automatically created with an empty `spec` when you start your Sensu backend.

Using the `Env` secrets provider may require you to synchronize environment variables in Sensu backend clusters. Read Use secrets management to learn how to configure the `Env` secrets provider.

YML

```
---
type: Env
api_version: secrets/v1
metadata:
  name: env
spec: {}
```

JSON

```
{
  "type": "Env",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "env"
  },
  "spec": {}
}
```

Secrets provider configuration

You can use the enterprise/secrets/v1 API endpoints to create, view, and manage your secrets provider configuration.

For example, to retrieve the list of secrets providers:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers \
-H "Authorization: Key $SENSU_API_KEY"
```

Secrets provider specification

***NOTE**: The attribute descriptions in this section use the `VaultProvider` datatype. Review the Env secrets provider example for an example definition for the `Env` datatype.*

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For secrets configuration in this version of Sensu, the api_version should always be `secrets/v1`.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `api_version: secrets/v1` **JSON** `{ "api_version": "secrets/v1" }`

metadata	
description	Top-level scope that contains the secrets provider `name` and

	`created_by` field. Namespace is not supported in the metadata because secrets providers are cluster-wide resources.
required	true
type	Map of key-value pairs
example	**YML** ```yaml metadata: name: vault created_by: admin ``` **JSON** ```json { "metadata": { "name": "vault", "created_by": "admin" } } ```

spec

description	Top-level map that includes secrets provider configuration spec attributes.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs
example	**YML** ```yaml spec: client: address: https://vaultserver.example.com:8200 max_retries: 2 rate_limiter: limit: 10 burst: 100 timeout: 20s ```

```
    tls:
      ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
    token: VAULT_TOKEN
    version: v1
```

JSON

```
{
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "max_retries": 2,
      "rate_limiter": {
        "limit": 10,
        "burst": 100
      },
      "timeout": "20s",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "token": "VAULT_TOKEN",
      "version": "v1"
    }
  }
}
```

type	
description	Top-level attribute that specifies the resource type. May be either `Env` (if you are using Sensu's secrets provider) or `VaultProvider` (if you are using HashiCorp Vault as the secrets provider).
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `type: VaultProvider`

JSON

```
{
  "type": "VaultProvider"
}
```

Metadata attributes

created_by	
description	Username of the Sensu user who created the secrets provider or last updated the secrets provider. Sensu automatically populates the `created_by` field when the secrets provider is created or updated.
required	false
type	String
example	**YML** `created_by: admin` **JSON** `{ "created_by": "admin" }`

name	
description	Provider name used internally by Sensu.
required	true

type	String

YML

example	

```
name: vault
```

JSON

```
{
  "name": "vault"
}
```

Spec attributes

client

description	Map that includes secrets provider configuration client attributes.
required	true
type	Map of key-value pairs

YML

example	

```
client:
  address: https://vaultserver.example.com:8200
  max_retries: 2
  rate_limiter:
    limit: 10
    burst: 100
  timeout: 20s
  tls:
    ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
  token: VAULT_TOKEN
  version: v1
```

JSON

```
{
```

```
  "client": {
    "address": "https://vaultserver.example.com:8200",
    "max_retries": 2,
    "rate_limiter": {
      "limit": 10,
      "burst": 100
    },
    "timeout": "20s",
    "tls": {
      "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
    },
    "token": "VAULT_TOKEN",
    "version": "v1"
  }
}
```

Client attributes

address	
description	Vault server address.
required	true
type	String
example	**YML** `address: https://vaultserver.example.com:8200` **JSON** `{ "address": "https://vaultserver.example.com:8200" }`

max_retries

description	Number of times to retry connecting to the Vault provider.
required	true
type	Integer
default	2
example	**YML** `max_retries: 2` **JSON** `{ "max_retries": 2 }`

rate_limiter

description	Maximum rate and burst limits for the enterprise/secrets/v1 API endpoint. Read rate_limiter attributes for more information.
required	false
type	Map of key-value pairs
example	**YML** `rate_limiter:` `  limit: 10` `  burst: 100` **JSON** `{ "rate_limiter": {`

```
    "limit": 10,
    "burst": 100
  }
}
```

timeout

description	Provider connection timeout (hard stop).
required	false
type	String
default	60s
example	**YML** `timeout: 20s` **JSON** `{ "timeout": "20s" }`

tls

description	TLS object. Vault only works with TLS configured. You may need to set up a Certificate Authority (CA) certificate if it is not already stored in your operating system's trust store. To do this, set the TLS object and provide the `ca_cert` path. You may also need to set up `client_cert`, `client_key`, or `cname`.
required	false
type	Map of key-value pairs

example	**YML** ```yaml tls: ca_cert: "/etc/ssl/certs/vault_ca_cert.pem" client_cert: "/etc/ssl/certs/vault_cert.pem" client_key: "/etc/ssl/certs/vault_key.pem" cname: vault_client.example.com ``` **JSON** ```json { "tls": { "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem", "client_cert": "/etc/ssl/certs/vault_cert.pem", "client_key": "/etc/ssl/certs/vault_key.pem", "cname": "vault_client.example.com" } } ```

token

description	Vault token to use for authentication.
required	true
type	String
example	**YML** ```yaml token: VAULT_TOKEN ``` **JSON** ```json { "token": "VAULT_TOKEN" } ```

version	
description	HashiCorp Vault key/value store version.
required	true
type	String
allowed values	`v1` and `v2`
example	**YML** `version: v1` **JSON** `{ "version": "v1" }`

Rate limiter attributes

burst	
description	Maximum amount of burst allowed in a rate interval for the enterprise/secrets/v1 API endpoint.
required	false
type	Integer
example	**YML** `burst: 100` **JSON** `{`

```
  "burst": 100
}
```

limit	
description	Maximum number of secrets requests per second that can be transmitted to the backend with the enterprise/secrets/v1 API endpoint.
required	false
type	Float
example	**YML** `limit: 10.0` **JSON** `{ "limit": 10.0 }`

Guides Index

This index links to every guide in the Sensu documentation. Guides describe how to configure Sensu to complete specific observability tasks and workflows to monitor server resources, route alerts and reduce alert fatigue, export metrics, plan maintenance windows, and more, with examples and step-by-step walkthroughs.

- Scale Sensu Go with Enterprise datastore
- Secure PostgreSQL
- Secure Sensu
- Send data to Sumo Logic with Sensu
- Send email alerts with a pipeline
- Send PagerDuty alerts with Sensu
- Send Slack alerts with a pipeline
- Use API keys to authenticate to Sensu
- Use dynamic runtime assets to install plugins
- Use secrets management in Sensu

Sensuctl CLI

Sensuctl is the command line tool for managing resources within Sensu. It works by calling Sensu's underlying API to create, read, update, and delete events, entities, and resources.

Sensuctl is available for Linux, macOS, and Windows. For Windows operating systems, sensuctl uses `cmd.exe` for the execution environment. For all other operating systems, sensuctl uses the Bourne shell (sh).

Read Install Sensu to install and configure sensuctl.

First-time setup and authentication

To log in to sensuctl and connect to the Sensu backend by following interactive prompts, run:

```
sensuctl configure
```

The `sensuctl configure` command starts the prompts for interactive setup. The first prompt is for the authentication method you wish to use: username/password or OIDC.

Sensuctl uses your username and password or OIDC credentials to obtain access and refresh tokens via the Sensu /auth API. The access and refresh tokens are HMAC-SHA256 JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature.

Upon successful authentication, sensuctl stores the access and refresh tokens in a `cluster` configuration file under the current user's home directory. For example, on Unix systems, sensuctl stores the tokens in `$HOME/.config/sensu/sensuctl/cluster`.

The `sensuctl configure` interactive prompts require you to select an authentication method and enter the Sensu backend URL, namespace, and preferred output format.

Username/password authentication

If you select username/password authentication, you will be prompted to enter your username and password Sensu access credentials.

Username/password authentication applies to the following authentication providers:

- Built-in basic authentication
- Lightweight Directory Access Protocol (LDAP) authentication (commercial feature)
- Active Directory (AD) authentication (commercial feature)

This example shows the `sensuctl configure` interactive dialog for the username/password authentication method:

```
Authentication method: username/password
Sensu Backend API URL: http://127.0.0.1:8080
Namespace: default
Preferred output format: tabular
Username: <YOUR_USERNAME>
Password: <YOUR_PASSWORD>
```

OIDC authentication

This example shows the `sensuctl configure` interactive dialog if you select the OIDC authentication method:

```
Authentication method: OIDC
Sensu Backend API URL: http://127.0.0.1:8080
Namespace: default
Preferred output format: tabular
Launching browser to complete the login via your OIDC provider at following URL:

  http://127.0.0.1:8080/api/enterprise/authentication/v2/oidc/authorize?
callback=http%3A%2F%2Flocalhost%3A8000%2Fcallback

You may also manually open this URL. Waiting for callback...
```

If you are using a desktop, a browser should open to allow you to authenticate and log in via your

OIDC provider.If a browser does not open, launch a browser and go to the OIDC URL listed at the end of the `sensuctl configure` interactive dialog to complete authentication and log in via your OIDC provider.

> ***NOTE**: You can also use `sensuctl login oidc` to log in to sensuctl with OIDC.*

Use flags to configure sensuctl in non-interactive mode

Run `sensuctl configure` non-interactively by adding the `-n` (`--non-interactive`) flag.For example, the following command configures sensuctl with the same values used in the username/password interactive example:

```
sensuctl configure -n --url http://127.0.0.1:8080 --format tabular --username
<YOUR_USERNAME> --password '<YOUR_PASSWORD>'
```

Run `sensuctl configure -h` to view command-specific and global flags that you can use to set up sensuctl when you bypass interactive mode:

```
Initialize sensuctl configuration

Usage:  sensuctl configure [flags]

Flags:
      --format string     preferred output format (default "tabular")
  -h, --help              help for configure
  -n, --non-interactive   do not administer interactive questionnaire
      --oidc              use an OIDC provider for authentication
      --password string   password
      --port int          port for local HTTP web server used for OAuth 2 callback
during OIDC authentication (default 8000)
      --url string        the sensu backend url (default "http://localhost:8080")
      --username string   username

Global Flags:
      --api-key string             API key to use for authentication
      --api-url string             host URL of Sensu installation
```

```
      --cache-dir string           path to directory containing cache & temporary
files (default "/Users/hillaryfraley/Library/Caches/sensu/sensuctl")
      --config-dir string          path to directory containing configuration files
(default "/Users/hillaryfraley/.config/sensu/sensuctl")
      --insecure-skip-tls-verify   skip TLS certificate verification (not
recommended!)
      --namespace string           namespace in which we perform actions (default
"default")
      --timeout duration           timeout when communicating with sensu backend
(default 15s)
      --trusted-ca-file string     TLS CA certificate bundle in PEM format
```

Username, password, and namespace

The Sensu backend installation process creates an administrator username and password and a `default` namespace.

> ***NOTE**: For a **new** installation, you can set administrator credentials with environment variables during initialization. If you are using Docker and you do not include the environment variables to set administrator credentials, the backend will initialize with the default username (`admin`) and password (`P@ssw0rd!`).*

Your ability to get, list, create, update, and delete resources with sensuctl depends on the permissions assigned to your Sensu user. For more information about configuring Sensu access control, read the role-based access control (RBAC) reference.

Change the admin user's password

After you configure sensuctl and authenticate, you can change the admin user's password. Run:

```
sensuctl user change-password --interactive
```

You must specify the user's current password to use the `sensuctl user change-password` command.

Reset a user password

To reset a user password without specifying the current password, run:

```
sensuctl user reset-password <USERNAME> --interactive
```

You must have admin permissions to use the `sensuctl user reset-password` command.

Test a user password

To test the password for a user created with Sensu's built-in basic authentication:

```
sensuctl user test-creds <USERNAME> --password 'password'
```

An empty response indicates valid credentials. A `request-unauthorized` response indicates invalid credentials.

***NOTE**: The `sensuctl user test-creds` command tests passwords for users created with Sensu's built-in basic authentication. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).*

For example, if you test LDAP credentials with the `sensuctl user test-creds` command, the backend will log an error, even if the LDAP credentials are correct:

```
{"component":"apid.routers","error":"basic provider is
disabled","level":"info","msg":"invalid username and/or password","time":"2020-02-
07T20:42:14Z","user":"dev"}
```

Generate a password hash

You can use a password hash instead of a user's password in the sensuctl commands to create and edit users. The `sensuctl user hash-password` command creates a bcrypt hash of the specified

password.

To generate a password hash for a specified cleartext password, run:

```
sensuctl user hash-password <PASSWORD>
```

Sensu backend URL

The Sensu backend URL is the HTTP or HTTPS URL where sensuctl can connect to the Sensu backend server. The default URL is `http://127.0.0.1:8080`.

To connect to a Sensu cluster, connect sensuctl to any single backend in the cluster. For information about configuring the Sensu backend URL, read the backend reference.

Preferred output format

After you configure sensuctl, you can change the default output format for sensuctl responses. Sensuctl supports the following output formats:

Format	Description
`tabular`	Output is organized in user-friendly columns. Tabular is the default output format.
`yaml`	Output is in YAML format. Resource definitions include the resource `type` and `api_version` as well as an outer-level `spec` "wrapping" for the resource attributes.
`wrapped-json`	Output is in JSON format. Resource definitions include the resource `type` and `api_version` as well as an outer-level `spec` "wrapping" for the resource attributes.
`json`	Output is in JSON format. Resource definitions **do not** include the resource `type` and `api_version` or an outer-level `spec` "wrapping".

Use `sensuctl config set-format` to change the preferred output format.

Output format significance

To use sensuctl create to create a resource, you must provide the resource definition in `yaml` or `wrapped-json` format. These formats include the resource `type`, which sensuctl needs to determine what kind of resource to create.

The Sensu API uses `json` output format for responses for APIs in the `core` group. For APIs that are not in the `core` group, responses are in the `wrapped-json` output format.

Sensu sends events to the backend in `json` format, without the `spec` attribute wrapper or `type` and `api_version` attributes.

Sensuctl configuration files

During configuration, sensuctl creates configuration files that contain information for connecting to your Sensu Go deployment. You can find these files at `$HOME/.config/sensu/sensuctl/profile` and `$HOME/.config/sensu/sensuctl/cluster`.

Use the `cat` command to view the contents of the configuration files. For example, to view your sensuctl profile configuration, run:

```
cat .config/sensu/sensuctl/profile
```

The response should be similar to this example:

```
{
  "format": "tabular",
  "namespace": "default",
  "username": "admin"
}
```

To view your sensuctl cluster configuration, run:

```
cat .config/sensu/sensuctl/cluster
```

The response should be similar to this example:

```
{
  "api-url": "http://localhost:8080",
  "trusted-ca-file": "",
  "insecure-skip-tls-verify": false,
  "access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "expires_at": 1550082282,
  "refresh_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

The sensuctl configuration files are useful if you want to know which cluster you're connecting to or which namespace or username you're currently configured to use.

Get help for sensuctl commands

Sensuctl supports a `--help` flag for each command and subcommand. The help response includes a usage template and lists of any available flags and further commands and subcommands.

To list global and command-specific flags for sensuctl in general, run:

```
sensuctl --help
```

To list available flags and subcommands for a sensuctl command like `sensuctl check` or `sensuctl create`, run:

```
sensuctl check --help
```

```
sensuctl create --help
```

To list available flags for a complete sensuctl command like `sensuctl check delete`, run:

```
sensuctl check delete --help
```

Manage sensuctl

Use the `sencutl config` command to view and modify the current sensuctl configuration.

To view flags and command options, run:

```
sensuctl config --help
```

The response lists the global flags and commands available to use with `sensuctl config`:

```
Modify sensuctl configuration

Usage:  sensuctl config COMMAND

Flags:
  -h, --help   help for config

Global Flags:
      --api-key string             API key to use for authentication
      --api-url string             host URL of Sensu installation
      --cache-dir string           path to directory containing cache & temporary
files (default "/home/vagrant/.cache/sensu/sensuctl")
      --config-dir string          path to directory containing configuration files
(default "/home/vagrant/.config/sensu/sensuctl")
      --insecure-skip-tls-verify   skip TLS certificate verification (not
recommended!)
      --namespace string           namespace in which we perform actions (default
"default")
      --timeout duration           timeout when communicating with sensu backend
(default 15s)
      --trusted-ca-file string     TLS CA certificate bundle in PEM format

Commands:
  set-format    Set format for active profile
```

```
  set-namespace Set namespace for active profile
  set-timeout   Set timeout for active profile in duration format (ex: 15s)
  view          Display active configuration
```

There are also commands for logging out of sensuctl and viewing the current sensuctl version.

View sensuctl config

To view the active configuration for sensuctl:

```
sensuctl config view
```

The `sensuctl config view` response includes the Sensu backend URL, default namespace for the current user, default output format for the current user, and currently configured username:

```
=== Active Configuration
API URL:   http://127.0.0.1:8080
Namespace: default
Format:    tabular
Username:  admin
```

Set preferred output format

Use the `set-format` command to change the preferred output format for the current user.

For example, to change the default tabular format to YAML for all sensuctl commands, run:

```
sensuctl config set-format yaml
```

You can also use the `--format` flag to set the output format for the response to a single sensuctl command. For example, to keep the default format set at tabular, but retrieve a specific entity definition in YAML format, run:

```
sensuctl entity info <ENTITY_NAME> --format yaml
```

Set namespace

Use the `set-namespace` command to change the default namespace for the current user. For more information about configuring Sensu access control, read the RBAC reference.

For example, to change the default namespace to `development`:

```
sensuctl config set-namespace development
```

Log out of sensuctl

To log out of sensuctl:

```
sensuctl logout
```

To log back in to sensuctl:

```
sensuctl configure
```

View the sensuctl version number

To display the current version of sensuctl:

```
sensuctl version
```

Use global flags for sensuctl settings

Global flags modify settings specific to sensuctl, such as the Sensu backend URL and namespace.

```
--api-key string             API key to use for authentication
--api-url string             host URL of Sensu installation
--cache-dir string           path to directory containing cache & temporary files
(default "/home/vagrant/.cache/sensu/sensuctl")
--config-dir string          path to directory containing configuration files (default
"/home/vagrant/.config/sensu/sensuctl")
--insecure-skip-tls-verify   skip TLS certificate verification (not recommended!)
--namespace string           namespace in which we perform actions (default
"default")
--timeout duration           timeout when communicating with sensu backend (default
15s)
--trusted-ca-file string     TLS CA certificate bundle in PEM format
```

You can use global flags with most sensuctl commands. To set global flags permanently, edit `.config/sensu/sensuctl/{cluster, profile}`.

Use shell autocompletion with sensuctl

Use shell autocompletion to create and run valid sensuctl commands. After you install and configure autocompletion, you can use the **tab** key to view and select from available options for each part of a sensuctl command directly from the command line.

Type `sensuctl` and press **tab** to view the list of available variables:

```
api-key               cluster-role          configure             edit
handler               logout                role                   user
asset                 cluster-role-binding  create                 entity
help                  mutator               role-binding           version
auth                  command               delete                 env
hook                  namespace             secret
check                 completion            describe-type          event
license               pipeline              silenced
cluster               config                dump                  filter
login                 prune                  tessen
```

Type your selected variable and press **tab** again to view the list of available variables to complete the command:

```
create  delete  info    list
```

Type your selected variable to complete the command and press **enter** to run it.

Install and configure autocompletion for sensuctl

Follow the instructions in this section to install and configure Bash or zsh autocompletion for sensuctl.

Install and configure for Bash

To install and configure Bash autocompletion for sensuctl:

1. Install bash-completion.

   > ***NOTE**: If you use a current version of Linux in a non-minimal installation, bash-completion may already be installed.*

   To install bash-completion on macOS, run:

   ```
   brew install bash-completion
   ```

   Open `~/.bash_profile`, add the following lines, and save:

   ```
   if [ -f $(brew --prefix)/etc/bash_completion ]; then
   . $(brew --prefix)/etc/bash_completion
   fi
   ```

2. Open `~/.bash_profile`, add the following line, and save:

   ```
   ```

```
source <(sensuctl completion bash)
```

3. Run the following command to source your `~/.bash_profile` file so that its resources are available:

```
source ~/.bash_profile
```

Shell autocompletion should now be available for sensuctl.

Install and configure for zsh

To install and configure zsh autocompletion for sensuctl:

1. Open `~/.zshrc`, add the following line, and save:

```
source <(sensuctl completion zsh)
```

2. Run the following command to source your `~/.zshrc` file so that its resources are available:

```
source ~/.zshrc
```

Create and manage resources with sensuctl

Use the sensuctl command line tool to create and manage resources within Sensu.Sensuctl works by calling Sensu's underlying API to create, read, update, and delete resources, events, and entities.

Create resources

The `sensuctl create` command allows you to create or update resources by reading from STDIN or a file.

The `create` command accepts Sensu resource definitions in `yaml` or `wrapped-json` formats, which wrap the contents of the resource in `spec` and identify the resource `type` and `api_version`. Review the list of supported resource types `for sensuctl create`.Read the reference docs for information about creating resource definitions.

Resources that you create with `sensuctl create` will include the following label in the metadata:

YML

```
sensu.io/managed_by: sensuctl
```

JSON

```
{
  "sensu.io/managed_by": "sensuctl"
}
```

You can create more than one resource at a time with `sensuctl create`.If you use YAML, separate the resource definitions by a line with three hyphens: `---`.If you use wrapped JSON, separate the resources *without* a comma.

***NOTE**: You can also use the `sensuctl <RESORUCE_TYPE> create` command to create resources*

with sensuctl.Read Use the create subcommand for more information and an example.

Create resources from STDIN

The following example demonstrates how to use the EOF function with `sensuctl create` to create two resources by reading from STDIN: a `marketing-site` check and a `slack` handler.

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: marketing-site
spec:
  command: http-check -u https://sensu.io
  subscriptions:
  - demo
  interval: 15
  handlers:
  - slack
---
type: Handler
api_version: core/v2
metadata:
  name: slack
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXX
XXXXXXXXXX
  type: pipe
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
```

```
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata" : {
    "name": "marketing-site"
    },
  "spec": {
    "command": "http-check -u https://sensu.io",
    "subscriptions": ["demo"],
    "interval": 15,
    "handlers": ["slack"]
  }
}
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXXX"
    ],
    "type": "pipe"
  }
}
EOF
```

Create resources from a file

The following example demonstrates how to use the `--file` flag with `sensuctl create` to create a `marketing-site` check and a `slack` handler.

First, copy these resource definitions and save them in a file named `my-resources.yml` or `my-resources.json`:

YML

```
---
```

```
type: CheckConfig
api_version: core/v2
metadata:
  name: marketing-site
spec:
  command: http-check -u https://sensu.io
  subscriptions:
  - demo
  interval: 15
  handlers:
  - slack
---
type: Handler
api_version: core/v2
metadata:
  name: slack
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXX
XXXXXXXXXX
  type: pipe
```

SHELL

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata" : {
    "name": "marketing-site"
    },
  "spec": {
    "command": "http-check -u https://sensu.io",
    "subscriptions": ["demo"],
    "interval": 15,
    "handlers": ["slack"]
  }
}
{
  "type": "Handler",
```

```
  "api_version": "core/v2",
  "metadata": {
    "name": "slack"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
    ],
    "type": "pipe"
  }
}
```

Run the following command to create the resources from `my-resources.yml` or `my-resources.json`:

SHELL

```
sensuctl create --file my-resources.yml
```

SHELL

```
sensuctl create --file my-resources.json
```

Or:

SHELL

```
cat my-resources.yml | sensuctl create
```

SHELL

```
cat my-resources.json | sensuctl create
```

sensuctl create flags

Run `sensuctl create -h` to view a usage example with command-specific and global flags:

```
Create or replace resources from file or URL (path, file://, http[s]://), or STDIN
otherwise.

Usage:  sensuctl create [-r] [[-f URL] ... ] [flags]

Flags:
  -f, --file strings   Files, directories, or URLs to create resources from
  -h, --help           help for create
  -r, --recursive      Follow subdirectories

Global Flags:
      --api-key string             API key to use for authentication
      --api-url string             host URL of Sensu installation
      --cache-dir string           path to directory containing cache & temporary
files (default "/home/vagrant/.cache/sensu/sensuctl")
      --config-dir string          path to directory containing configuration files
(default "/home/vagrant/.config/sensu/sensuctl")
      --insecure-skip-tls-verify   skip TLS certificate verification (not
recommended!)
      --namespace string           namespace in which we perform actions (default
"default")
      --timeout duration           timeout when communicating with sensu backend
(default 15s)
      --trusted-ca-file string     TLS CA certificate bundle in PEM format
```

sensuctl create resource types

Use sensuctl create with any of the following resource types:

sensuctl create types			
ad	AdhocRequest	Asset	CheckConfig
ClusterRole	ClusterRoleBinding	Entity	Env

`EtcdReplicators`	`Event`	`EventFilter`	`GlobalConfig`
`Handler`	`HookConfig`	`ldap`	`Mutator`
`Namespace`	`oidc`	`PostgresConfig`	`Role`
`RoleBinding`	`Secret`	`Silenced`	`SumoLogicMetricsHandler`
`TCPStreamHandler`	`TessenConfig`	`User`	`VaultProvider`

Create resources across namespaces

If you omit the `namespace` attribute from resource definitions, you can use the `senusctl create --namespace` flag to specify the namespace for a group of resources at the time of creation. This allows you to replicate resources across namespaces without manual editing.

To learn more about namespaces, read the namespaces reference. The RBAC reference includes a list of namespaced resource types.

The `sensuctl create` command applies namespaces to resources in the following order, from highest precedence to lowest:

1. **Namespace specified within resource definitions**: You can specify a resource's namespace within individual resource definitions using the `namespace` attribute. Namespaces specified in resource definitions take precedence over all other methods.
2. **`--namespace` flag**: If resource definitions do not specify a namespace, Sensu applies the namespace provided by the `sensuctl create --namespace` flag.
3. **Current sensuctl namespace configuration**: If you do not specify an embedded `namespace` attribute or use the `--namespace` flag, Sensu applies the namespace configured in the current sensuctl session. Read Manage sensuctl to view your current session config and set the session namespace.

For example, this handler does not include a `namespace` attribute:

YML

```
---
type: Handler
api_version: core/v2
```

```
metadata:
  name: pagerduty
spec:
  command: sensu-pagerduty-handler
  env_vars:
  - PAGERDUTY_TOKEN=SECRET
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "pagerduty"
  },
  "spec": {
    "command": "sensu-pagerduty-handler",
    "env_vars": [
      "PAGERDUTY_TOKEN=SECRET"
    ],
    "type": "pipe"
  }
}
```

If you save this resource definition in a file named `pagerduty.yml` or `pagerduty.json`, you can create the `pagerduty` handler in any namespace with specific sensuctl commands.

To create the handler in the `default` namespace:

SHELL

```
sensuctl create --file pagerduty.yml --namespace default
```

SHELL

```
sensuctl create --file pagerduty.json --namespace default
```

To create the `pagerduty` handler in the `production` namespace:

SHELL

```
sensuctl create --file pagerduty.yml --namespace production
```

SHELL

```
sensuctl create --file pagerduty.json --namespace production
```

To create the `pagerduty` handler in the current session namespace:

SHELL

```
sensuctl create --file pagerduty.yml
```

SHELL

```
sensuctl create --file pagerduty.json
```

Delete resources

The `sensuctl delete` command allows you to delete resources by reading from STDIN or a file.

You can use `sensuctl delete` with the same resource types as `sensuctl create`.

The `delete` command accepts Sensu resource definitions in `wrapped-json` and `yaml` formats. To be deleted successfully, the name and namespace of a resource provided to the `delete` command must match the name and namespace of an existing resource.

Delete resources with STDIN

To delete the `marketing-site` check from the current namespace with STDIN, run:

SHELL

```
cat << EOF | sensuctl delete
```

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: marketing-site
spec:
  command: http-check -u https://sensu.io
  subscriptions:
  - demo
  interval: 15
  handlers:
  - slack
EOF
```

SHELL

```
cat << EOF | sensuctl delete
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata" : {
    "name": "marketing-site"
    },
  "spec": {
    "command": "http-check -u https://sensu.io",
    "subscriptions": ["demo"],
    "interval": 15,
    "handlers": ["slack"]
  }
}
EOF
```

Delete resources using a file

To delete all resources listed in a specific file from Sensu (in this example, a file named `my-resources.yml` or `my-resources.json`):

SHELL

```
sensuctl delete --file my-resources.yml
```

SHELL

```
sensuctl delete --file my-resources.json
```

Or:

SHELL

```
cat my-resources.yml | sensuctl delete
```

SHELL

```
cat my-resources.json | sensuctl delete
```

sensuctl delete flags

Run `sensuctl delete -h` to view a usage example with command-specific and global flags:

```
Delete resources from file or STDIN

Usage:  sensuctl delete [-f FILE] [flags]

Flags:
  -f, --file string   File to delete resources from
  -h, --help          help for delete

Global Flags:
      --api-key string             API key to use for authentication
      --api-url string             host URL of Sensu installation
      --cache-dir string           path to directory containing cache & temporary
files (default "/home/vagrant/.cache/sensu/sensuctl")
      --config-dir string          path to directory containing configuration files
(default "/home/vagrant/.config/sensu/sensuctl")
      --insecure-skip-tls-verify   skip TLS certificate verification (not
recommended!)
      --namespace string           namespace in which we perform actions (default
```

```
"default")
      --timeout duration           timeout when communicating with sensu backend
(default 15s)
      --trusted-ca-file string     TLS CA certificate bundle in PEM format
```

Delete resources across namespaces

To use the `senusctl delete --namespace` flag to specify the namespace for a group of resources at the time of deletion, omit the `namespace` attribute from resource definitions. This allows you to remove resources across namespaces without manual editing.

For example, suppose you added the `pagerduty` handler from Create resources across namespaces in every namespace. To delete the `pagerduty` handler from only the `production` namespace using STDIN, run:

SHELL

```
cat << EOF | sensuctl delete --namespace production
---
type: Handler
api_version: core/v2
metadata:
  name: pagerduty
spec:
  command: sensu-pagerduty-handler
  env_vars:
  - PAGERDUTY_TOKEN=SECRET
  type: pipe
EOF
```

SHELL

```
cat << EOF | sensuctl delete --namespace production
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "pagerduty"
  },
  "spec": {
```

```
    "command": "sensu-pagerduty-handler",
    "env_vars": [
      "PAGERDUTY_TOKEN=SECRET"
    ],
    "type": "pipe"
  }
}
EOF
```

You can also use the `sensuctl delete` command with a file that includes the `pagerduty` handler definition (in these examples, the file name is `pagerduty.yml` or `pagerduty.json`).

Delete the `pagerduty` handler from the `default` namespace with this command:

SHELL

```
sensuctl delete --file pagerduty.yml --namespace default
```

SHELL

```
sensuctl delete --file pagerduty.json --namespace default
```

To delete the `pagerduty` handler from the `production` namespace:

SHELL

```
sensuctl delete --file pagerduty.yml --namespace production
```

SHELL

```
sensuctl delete --file pagerduty.json --namespace production
```

To delete the `pagerduty` handler in the current session namespace:

SHELL

```
sensuctl delete --file pagerduty.yml
```

SHELL

```
sensuctl delete --file pagerduty.json
```

Update resources

Sensuctl allows you to update resource definitions with a text editor. To use `sensuctl edit`, specify the resource type and resource name.

> ***NOTE**: You cannot use sensuctl to update agent-managed entities. Requests to update agent-managed entities via sensuctl will fail and return an error.*

For example, to edit a handler named `slack` with `sensuctl edit`:

```
sensuctl edit handler slack
```

> ***NOTE**: You cannot use sensuctl to update agent-managed entities. Requests to update agent-managed entities via sensuctl will fail and return an error.*

sensuctl edit flags

Run `sensuctl edit -h` to view a usage example with command-specific and global flags:

```
Edit resources interactively

Usage:  sensuctl edit [RESOURCE TYPE] [KEY]... [flags]

Flags:
  -b, --blank           edit a blank resource, and create it on save
      --format string   format of data returned ("json"|"wrapped-
json"|"tabular"|"yaml") (default "tabular")
  -h, --help            help for edit
```

```
Global Flags:
      --api-key string             API key to use for authentication
      --api-url string             host URL of Sensu installation
      --cache-dir string           path to directory containing cache & temporary
files (default "/home/vagrant/.cache/sensu/sensuctl")
      --config-dir string          path to directory containing configuration files
(default "/home/vagrant/.config/sensu/sensuctl")
      --insecure-skip-tls-verify   skip TLS certificate verification (not
recommended!)
      --namespace string           namespace in which we perform actions (default
"default")
      --timeout duration           timeout when communicating with sensu backend
(default 15s)
      --trusted-ca-file string     TLS CA certificate bundle in PEM format
```

sensuctl edit resource types

Use the `sensuctl edit` command with any of the following resource types:

sensuctl edit types			
`asset`	`auth`	`check`	`cluster`
`cluster-role`	`cluster-role-binding`	`entity`	`event`
`filter`	`handler`	`hook`	`mutator`
`namespace`	`pipeline`	`role`	`role-binding`
`silenced`	`user`		

Manage resources

Sensuctl provides the commands listed below for managing individual Sensu resources.Combine the resource command with a subcommand to complete operations like listing all checks or deleting a specific silence.

- `sensuctl asset`
- `sensuctl auth` (commercial feature)
- `sensuctl check`
- `sensuctl cluster`
- `sensuctl cluster-role`
- `sensuctl cluster-role-binding`
- `sensuctl entity`
- `sensuctl event`
- `sensuctl filter`
- `sensuctl handler`
- `sensuctl hook`
- `sensuctl license` (commercial feature)
- `sensuctl mutator`
- `sensuctl namespace`
- `sensuctl pipeline`
- `sensuctl role`
- `sensuctl role-binding`
- `sensuctl secret`
- `sensuctl silenced`
- `sensuctl tessen`
- `sensuctl user`

Subcommands

Sensuctl provides a set of operation subcommands for each resource type.

To view the supported subcommands for a resource type, run the resource command followed by the help flag, `-h`. For example, to view the subcommands for `sensuctl check`, run:

```
sensuctl check -h
```

The response includes a usage example, the supported command-specific and global flags, and a list of supported subcommands.

Many resource types include a standard set of list, info, and delete operation subcommands:

```
delete                      delete resource given resource name
info                        show detailed resource information given resource name
list                        list resources
```

> ***NOTE**: The delete, info, and list subcommands are not supported for all resource types. Run `sensuctl <RESOURCE_TYPE> -h` to confirm which subcommands are supported for a specific resource type. You can also configure shell completion for sensuctl to view available variables for sensuctl commands.*

Use the commands with their flags and subcommands to get more information about your resources. For example, to list all monitoring checks:

```
sensuctl check list
```

To list checks from all namespaces:

```
sensuctl check list --all-namespaces
```

To write all checks to `my-resources.yml` in `yaml` format or to `my-resources.json` in `wrapped-json` format:

SHELL

```
sensuctl check list --format yaml > my-resources.yml
```

SHELL

```
sensuctl check list --format wrapped-json > my-resources.json
```

To view the definition for a check named `check-cpu`:

SHELL

```
sensuctl check info check-cpu --format yaml
```

SHELL

```
sensuctl check info check-cpu --format wrapped-json
```

To delete the definition for a check named `check-cpu`:

```
sensuctl check delete check-cpu
```

In addition to the delete, info, and list operations, many commands support flags and subcommands that allow you to take special action based on the resource type. The sections below describe some of the resource-specific operations.

Run `sensuctl <RESOURCE_TYPE> -h` to retrieve a complete list of the supported flags and subcommands for a specific resource command. You can also configure shell completion for sensuctl to view available variables for sensuctl commands.

Use the create subcommand

Many resource types include a `create` subcommand that you can use to create resources using flags. Run `sensuctl <RESOURCE_TYPE> create -h` to get a list of the supported flags for the resource type.

For example, run this command to create a check, using flags to specify the check's command, interval, subscriptions, and runtime assets:

```
sensuctl check create check_cpu \
--command 'check-cpu-usage -w 75 -c 90' \
--interval 60 \
--subscriptions system \
--runtime-assets sensu/check-cpu-usage
```

The command creates a check with the following definition:

YML

```
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: check_cpu
  namespace: default
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  pipelines: []
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu/check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
```

```
    "created_by": "admin",
    "name": "check_cpu",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "pipelines": [],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu/check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```


***NOTE**: Resources created with the `sensuctl <RESOURCE_TYPE> create` subcommand **do not** include the label `sensu.io/managed_by: sensuctl`.*

Handle large datasets

When using sensuctl to retrieve large datasets with the `list` command, add the `--chunk-size` flag

to prevent query timeouts and improve performance. The `--chunk-size` flag allows you to specify how many events Sensu should retrieve with each query. Sensu will make a series of queries to retrieve all resources instead of a single query.

For example, the following command returns the same output as `sensuctl event list` but makes multiple API queries, each for the number of resources specified with `--chunk-size`, instead of one query for the complete dataset:

```
sensuctl event list --chunk-size 500
```

Execute a check on demand

The `sensuctl check execute` command executes the specified check on demand:

```
sensuctl check execute <CHECK_NAME>
```

For example, the following command executes the `check-cpu` check with an attached message:

```
sensuctl check execute check-cpu --reason "giving a sensuctl demo"
```

You can also use the `--subscriptions` flag to override the subscriptions in the check definition:

```
sensuctl check execute check-cpu --subscriptions demo,webserver
```

Manage a Sensu cluster

The `sensuctl cluster` command lets you manage a Sensu cluster with the following subcommands:

```
health         get sensu health status
id             show sensu cluster id
member-add     add cluster member to an existing cluster, with comma-separated peer
```

```
addresses
member-list    list cluster members
member-remove  remove cluster member by ID
member-update  update cluster member by ID with comma-separated peer addresses
```

To view cluster members:

```
sensuctl cluster member-list
```

To review the health of your Sensu cluster:

```
sensuctl cluster health
```

Manually resolve events

Use `sensuctl event resolve` to manually resolve events:

```
sensuctl event resolve <ENTITY_NAME> <CHECK_NAME>
```

For example, the following command manually resolves an event created by the entity `webserver1` and the check `check-http`:

```
sensuctl event resolve webserver1 check-http
```

Use the sensuctl namespace command

The sensuctl namespace commands have a few special characteristics that you should be aware of.

sensuctl namespace create

Namespace names can contain alphanumeric characters and hyphens and must begin and end with an alphanumeric character.

senscutl namespace list

In the packaged Sensu Go distribution, `sensuctl namespace list` lists only the namespaces for which the current user has access.

sensuctl namespace delete

Namespaces must be empty before you can delete them. If the response to `sensuctl namespace delete` is `Error: resource is invalid: namespace is not empty`, the namespace may still contain events or other resources.

To remove all resources and events so that you can delete a namespace, run this command (replace `<NAMESPACE_NAME>` with the namespace you want to empty):

```
sensuctl dump entities,events,assets,checks,filters,handlers,secrets/v1.Secret --
namespace <NAMESPACE_NAME> | sensuctl delete
```

Prune resources

***COMMERCIAL FEATURE**: Access sensuctl pruning in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The `sensuctl prune` subcommand allows you to delete resources that do not appear in a given set of Sensu objects (called a "configuration") from a from a file, URL, or stdin. For example, you can use `sensuctl create` to to apply a new configuration, then use `sensuctl prune` to prune unneeded resources, resources that were created by a specific user or that include a specific label selector, and more.

***NOTE**: `sensuctl prune` is an alpha feature and may include breaking changes.*

`sensuctl prune` can only delete resources that have the label `sensu.io/managed_by: sensuctl`, which Sensu automatically adds to resources created with the `sensuctl create` command. This means you can only use `sensuctl prune` to delete resources that were created with `sensuctl create`.

The pruning operation always follows the role-based access control (RBAC) permissions of the current

user. For example, to prune resources in the `dev` namespace, the current user who sends the prune command must have delete access to the `dev` namespace.

Supported resource types

To retrieve the supported `sensuctl prune` resource types, run:

```
sensuctl describe-type all
```

The response will list all supported `sensuctl prune` resource types:

```
      Fully Qualified Name          Short Name       API Version          Type          Namespaced
_____________________________________ _____________ ____________________ ________________ ____________

authentication/v2.Provider            authentication/v2 Provider             false
licensing/v2.LicenseFile              licensing/v2      LicenseFile          false
store/v1.PostgresConfig               store/v1          PostgresConfig       false
federation/v1.EtcdReplicator          federation/v1     EtcdReplicator       false
federation/v1.Cluster                 federation/v1     Cluster              false
secrets/v1.Secret                     secrets/v1        Secret               true
secrets/v1.Provider                   secrets/v1        Provider             false
searches/v1.Search                    searches/v1       Search               true
web/v1.GlobalConfig                   web/v1            GlobalConfig         false
bsm/v1.RuleTemplate                   bsm/v1            RuleTemplate         true
bsm/v1.ServiceComponent               bsm/v1            ServiceComponent     true
pipeline/v1.SumoLogicMetricsHandler   pipeline/v1       SumoLogicMetricsHandler true
pipeline/v1.TCPStreamHandler          pipeline/v1       TCPStreamHandler     true
core/v2.Namespace          namespaces       core/v2           Namespace            false
core/v2.ClusterRole        clusterroles     core/v2           ClusterRole          false
core/v2.ClusterRoleBinding clusterrolebindings core/v2        ClusterRoleBinding   false
core/v2.User               users            core/v2           User                 false
core/v2.APIKey             apikeys          core/v2           APIKey               false
core/v2.TessenConfig       tessen           core/v2           TessenConfig         false
core/v2.Asset              assets           core/v2           Asset                true
core/v2.CheckConfig        checks           core/v2           CheckConfig          true
core/v2.Entity             entities         core/v2           Entity               true
core/v2.Event              events           core/v2           Event                true
core/v2.EventFilter        filters          core/v2           EventFilter          true
```

```
core/v2.Handler          handlers          core/v2          Handler          true
core/v2.HookConfig       hooks             core/v2          HookConfig       true
core/v2.Mutator          mutators          core/v2          Mutator          true
core/v2.Pipeline         pipelines         core/v2          Pipeline         true
core/v2.Role             roles             core/v2          Role             true
core/v2.RoleBinding      rolebindings      core/v2          RoleBinding      true
core/v2.Silenced         silenced          core/v2          Silenced         true
```

***NOTE**: Short names are only supported for core/v2 resources.*

sensuctl prune flags

Run `sensuctl prune -h` to view command-specific and global flags. The following table describes the command-specific flags.

Flag	Function and important notes
`-a` or `--all-users`	Prunes resources created by all users. Mutually exclusive with the `--users` flag. Defaults to false.
`-c` or `--cluster-wide`	Prunes any cluster-wide (non-namespaced) resources that are not defined in the configuration. Defaults to false.
`-d` or `--dry-run`	Prints the resources that will be pruned but does not actually delete them. Defaults to false.
`-f` or `--file`	Files, URLs, or directories to prune resources from. Strings.
`-h` or `--help`	Help for the prune command.
`--label-selector`	Prunes only resources that match the specified labels (comma-separated strings). Labels are a commercial feature.
`-o` or `--omit`	Resources that should be excluded from being pruned.
`-r` or `--recursive`	Prune command will follow subdirectories.
`-u` or `--users`	Prunes only resources that were created by the specified users (comma-separated strings). Defaults to the currently configured sensuctl user.

sensuctl prune usage

```
sensuctl prune <RESOURCE_TYPE>,<RESOURCE_TYPE>... -f <FILE_OR_URL> [-r] ... ] --
namespace <NAMESPACE> <FLAGS>
```

In this example `sensuctl prune` command:

- Replace `<RESOURCE_TYPE>` with the fully qualified name or short name of the resource you want to prune. You must specify at least one resource type or the `all` qualifier (to prune all resource types).
- Replace `<FILE_OR_URL>` with the name of the file or the URL that contains the set of Sensu objects you want to keep (the configuration).
- Replace `<NAMESPACE>` with the namespace where you want to apply pruning. If you omit the namespace qualifier, the command defaults to the current configured namespace.
- Replace `<FLAGS>` with the other flags you want to use, if any.

Use a comma separator to prune more than one resource in a single command. For example, to prune checks and dynamic runtime assets from the file `checks.yaml` or `checks.json` for the `dev` namespace and the `admin` and `ops` users:

SHELL

```
sensuctl prune core/v2.CheckConfig,core/v2.Asset --file checks.yaml --namespace dev --
users admin,ops
```

SHELL

```
sensuctl prune core/v2.CheckConfig,core/v2.Asset --file checks.json --namespace dev --
users admin,ops
```

`sensuctl prune` supports pruning resources by their fully qualified names or short names:

Fully qualified names:

```
sensuctl prune core/v2.CheckConfig,core/v2.Entity
```

Short names:

```
sensuctl prune checks,entities
```

Use the `all` qualifier to prune all supported resources:

```
sensuctl prune all
```

Use the `--omit` flag to identify resources you want to exclude from being pruned:

```
sensuctl prune all --omit
core/v2.Role,core/v2.RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBinding
```

Time formats

Sensuctl supports multiple formats for resource attributes that require a time. To specify an exact point in time (for example, when setting a silence), use full dates with times.

Although supported formats depend on the resource type, sensuctl generally supports the following formats for dates with time:

- RFC 3339 with numeric zone offset: `2018-05-10T07:04:00-08:00` or `2018-05-10T15:04:00Z`
- RFC 3339 with space delimiters and numeric zone offset: `2018-05-10 07:04:00 -08:00`
- Sensu alpha legacy format with canonical zone ID: `May 10 2018 7:04AM America/Vancouver`

Use the `--help` (`-h`) flag for specific sensuctl commands and resources to learn which time format to use.

Supported canonical time zone IDs are defined in the tz database.

***WARNING**: Windows does not support canonical zone IDs (for example, `America/Vancouver`).*

Back up and recover resources with sensuctl

The sensuctl dump command allows you to export your resources to standard out (stdout) or to a file. You can export all resources or a subset of them based on a list of resource types. The dump command supports exporting in `wrapped-json` and `yaml`.

For example, to export all resources for the current namespace to a file named `my-resources.yml` or `my-resources.json` in `yaml` or `wrapped-json` format:

SHELL

```
sensuctl dump all --format yaml --file my-resources.yml
```

SHELL

```
sensuctl dump all --format wrapped-json --file my-resources.json
```

You can restore exported resources with sensuctl create.

***NOTE**: The sensuctl dump command does not export user passwords — you must add the `password_hash` or `password` attribute to any exported users resources before restoring them with sensuctl create.*

In addition, sensuctl create does not restore API keys from a sensuctl dump backup, although you can use your backup as a reference for granting new API keys.

Because users and API keys require these additional steps to restore with sensuctl create, you might prefer to use the etcd snapshot and restore process as your primary backup and restore method. Take regular etcd snapshots and make regular sensuctl dump backups for extra reassurance.

Back up before a Sensu version upgrade

You should create a backup before you upgrade to a new version of Sensu. Here's the step-by-step process:

1. Create a backup folder.

```
mkdir backup
```

2. Create a backup of the entire cluster, except entities, events, and role-based access control (RBAC) resources, for all namespaces.

SHELL

```
sensuctl dump all \
--all-namespaces \
--omit
core/v2.Entity,core/v2.Event,core/v2.APIKey,core/v2.User,core/v2.Role,core/v2.
RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBinding \
--format yaml \
--file backup/config.yml
```

SHELL

```
sensuctl dump all \
--all-namespaces \
--omit
core/v2.Entity,core/v2.Event,core/v2.APIKey,core/v2.User,core/v2.Role,core/v2.
RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBinding \
--format wrapped-json \
--file backup/config.json
```

3. Export your RBAC resources, except API keys and users, for all namespaces.

SHELL

```
sensuctl dump
core/v2.Role,core/v2.RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBindin
g \
--all-namespaces \
```

```
--format yaml \
--file backup/rbac.yml
```

SHELL

```
sensuctl dump
core/v2.Role,core/v2.RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBindin
g \
--all-namespaces \
--format wrapped-json \
--file backup/rbac.json
```

4. Export your API keys and users resources for all namespaces.

SHELL

```
sensuctl dump core/v2.APIKey,core/v2.User \
--all-namespaces \
--format yaml \
--file backup/cannotrestore.yml
```

SHELL

```
sensuctl dump core/v2.APIKey,core/v2.User \
--all-namespaces \
--format wrapped-json \
--file backup/cannotrestore.json
```

***NOTE**: Passwords are not included when you export users. You must add the `password_hash` or `password` attribute to any exported `users` resources before you can use them with `sensuctl create`.*

Because users require this additional configuration and API keys cannot be restored from a sensuctl dump backup, consider exporting your API keys and users to a different folder than `backup`.

5. Export your entity resources for all namespaces (if desired).

SHELL

```
sensuctl dump core/v2.Entity \
--all-namespaces \
--format yaml \
--file backup/inventory.yml
```

SHELL

```
sensuctl dump core/v2.Entity \
--all-namespaces \
--format wrapped-json \
--file backup/inventory.json
```

> ***NOTE**: If you do not export your entities, proxy check requests will not be scheduled for the excluded proxy entities.*

Back up to populate new namespaces

You can create a backup copy of your existing resources with their namespaces stripped from the record. This backup allows you to replicate resources across namespaces without manual editing.

To create a backup of your resources that you can replicate in new namespaces:

1. Create a backup folder.

```
mkdir backup
```

2. Back up your pipeline resources for all namespaces, stripping namespaces so that your resources are portable for reuse in any namespace.

SHELL

```
sensuctl dump
core/v2.Asset,core/v2.CheckConfig,core/v2.HookConfig,core/v2.EventFilter,core/v2
```

```
.Mutator,core/v2.Handler,core/v2.Silenced,secrets/v1.Secret,secrets/v1.Provide
r \
--all-namespaces \
--format yaml | grep -v "^\s*namespace:" > backup/pipelines.yml
```

SHELL

```
sensuctl dump
core/v2.Asset,core/v2.CheckConfig,core/v2.HookConfig,core/v2.EventFilter,core/v2
.Mutator,core/v2.Handler,core/v2.Silenced,secrets/v1.Secret,secrets/v1.Provide
r \
--all-namespaces \
--format wrapped-json | grep -v "^\s*namespace:" > backup/pipelines.json
```

Restore resources from backup

When you are ready to restore your exported resources, use `sensuctl create`.

To restore everything you exported all at once, run:

```
sensuctl create -r -f backup/
```

To restore a subset of your exported resources (in this example, your RBAC resources), run:

SHELL

```
sensuctl create -f backup/rbac.yml
```

SHELL

```
sensuctl create -f backup/rbac.json
```

***NOTE**: When you export users, required password attributes are not included. You must add a `password_hash` or `password` to `users` resources before restoring them with the `sensuctl create` command.*

You can't restore API keys or users from a sensuctl dump backup. API keys must be reissued, but you can use your backup as a reference for granting new API keys to replace the exported keys.

Supported resource types

Use `sensuctl describe-type all` to retrieve the list of supported sensuctl dump resource types.

***NOTE**: Short names are only supported for core/v2 resources.*

```
sensuctl describe-type all
```

The response will list the names and other details for the supported resource types:

```
      Fully Qualified Name          Short Name       API Version          Type          Namespaced
─────────────────────────────────── ──────────── ─────────────────── ──────────────────── ────────────
  authentication/v2.Provider          authentication/v2   Provider             false
  licensing/v2.LicenseFile            licensing/v2        LicenseFile          false
  store/v1.PostgresConfig             store/v1            PostgresConfig       false
  federation/v1.EtcdReplicator        federation/v1       EtcdReplicator       false
  federation/v1.Cluster               federation/v1       Cluster              false
  secrets/v1.Secret                   secrets/v1          Secret               true
  secrets/v1.Provider                 secrets/v1          Provider             false
  searches/v1.Search                  searches/v1         Search               true
  web/v1.GlobalConfig                 web/v1              GlobalConfig         false
  bsm/v1.RuleTemplate                 bsm/v1              RuleTemplate         true
  bsm/v1.ServiceComponent             bsm/v1              ServiceComponent     true
  pipeline/v1.SumoLogicMetricsHandler pipeline/v1         SumoLogicMetricsHandler true
  pipeline/v1.TCPStreamHandler        pipeline/v1         TCPStreamHandler     true
  core/v2.Namespace                   namespaces          core/v2              Namespace            false
  core/v2.ClusterRole                 clusterroles        core/v2              ClusterRole          false
  core/v2.ClusterRoleBinding          clusterrolebindings core/v2              ClusterRoleBinding   false
  core/v2.User                        users               core/v2              User                 false
  core/v2.APIKey                      apikeys             core/v2              APIKey               false
  core/v2.TessenConfig                tessen              core/v2              TessenConfig         false
```

```
core/v2.Asset              assets          core/v2       Asset          true
core/v2.CheckConfig        checks          core/v2       CheckConfig    true
core/v2.Entity             entities        core/v2       Entity         true
core/v2.Event              events          core/v2       Event          true
core/v2.EventFilter        filters         core/v2       EventFilter    true
core/v2.Handler            handlers        core/v2       Handler        true
core/v2.HookConfig         hooks           core/v2       HookConfig     true
core/v2.Mutator            mutators        core/v2       Mutator        true
core/v2.Pipeline           pipelines       core/v2       Pipeline       true
core/v2.Role               roles           core/v2       Role           true
core/v2.RoleBinding        rolebindings    core/v2       RoleBinding    true
core/v2.Silenced           silenced        core/v2       Silenced       true
```

You can also list specific resource types by fully qualified name or short name:

```
sensuctl describe-type core/v2.CheckConfig
```

```
sensuctl describe-type checks
```

To list more than one type, use a comma-separated list:

```
sensuctl describe-type core/v2.CheckConfig,core/v2.EventFilter,core/v2.Handler
```

```
sensuctl describe-type checks,filters,handlers
```

Format the sensuctl describe-type response

Add the `--format` flag to specify how the resources should be formatted in the `sensuctl describe-type` response. The default is unformatted, but you can specify either `wrapped-json` or `yaml`:

SHELL

```
sensuctl describe-type core/v2.CheckConfig --format yaml
```

SHELL

```
sensuctl describe-type core/v2.CheckConfig --format wrapped-json
```

Example sensuctl dump commands

To export only checks for only the current namespace to stdout in YAML or wrapped JSON format:

SHELL

```
sensuctl dump core/v2.CheckConfig --format yaml
```

SHELL

```
sensuctl dump core/v2.CheckConfig --format wrapped-json
```

To export only handlers and filters for only the current namespace to a file named `my-handlers-and-filters` in YAML or wrapped JSON format:

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format yaml --file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format wrapped-json --file my-handlers-and-filters.json
```

To export resources for **all namespaces**, add the `--all-namespaces` flag to any sensuctl dump command. For example:

SHELL

```
sensuctl dump all --all-namespaces --format yaml --file my-resources.yml
```

SHELL

```
sensuctl dump all --all-namespaces --format wrapped-json --file my-resources.json
```

SHELL

```
sensuctl dump core/v2.CheckConfig --all-namespaces --format yaml
```

SHELL

```
sensuctl dump core/v2.CheckConfig --all-namespaces --format wrapped-json
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --all-namespaces --format yaml --file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --all-namespaces --format wrapped-json --file my-handlers-and-filters.json
```

You can use fully qualified names or short names to specify resources in sensuctl dump commands. Here's an example that uses fully qualified names:

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format yaml --file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format wrapped-json --file my-handlers-and-filters.json
```

Here’s an example that uses short names:

SHELL

```
sensuctl dump handlers,filters --format yaml --file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump handlers,filters --format wrapped-json --file my-handlers-and-
filters.json
```

Best practices for sensuctl dump

To reduce the running time for the sensuctl dump command, omit events and export only one namespace at a time.

Omit events from your sensuctl dump command to reduce the size of the exported payload and the system resources required to export. The most important part of a backup is capturing the Sensu configuration, and even with regular backups, events are likely to be outdated by the time you restore them. If you need access to all events, send them to a database store instead of including events in routine Sensu backups.

It takes longer to export resources from all namespaces at once than the resources from one namespace, especially as the number of resources in each namespace grows. To export resources more quickly, export a single namespace at a time.

Filter responses with sensuctl

***COMMERCIAL FEATURE**: Access sensuctl response filtering in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

Sensuctl supports response filtering for all commands using the `list` verb.For information about response filtering methods and available label and field selectors, read API response filtering.

Sensuctl-specific syntax

You can use the same methods, selectors, and examples for sensuctl response filtering as for API response filtering, except you'll format your requests with the `--label-selector` and `--field-selector` flags instead of cURL.

The standard sensuctl response filtering syntax is:

```
sensuctl <resource_type> list --<selector> '<filter_statement>'
```

To create a sensuctl response filtering command:

- Replace `<resource_type>` with the resource your filter is based on.
- Replace `<selector>` with either `label-selector` or `field-selector`, depending on which selector you want to use.
- Replace `<filter_statement>` with the filter to apply.

For example:

```
sensuctl event list --field-selector 'linux notin event.entity.subscriptions'
```

Sensuctl response filtering commands will also work with a single equals sign between the selector flag and the filter statement:

```
sensuctl event list --field-selector='linux notin event.entity.subscriptions'
```

The examples demonstrate how to construct sensuctl filter statements for different selectors and operators.

Filter operators

Sensuctl response filtering supports two equality-based operators, two set-based operators, one substring matching operator, and one logical operator.

operator	description	example
`==`	Equality	`check.publish == true`
`!=`	Inequality	`check.namespace != "default"`
`in`	Included in	`linux in check.subscriptions`
`notin`	Not included in	`slack notin check.handlers`
`matches`	Substring matching	`check.name matches "linux-"`
`&&`	Logical AND	`check.publish == true && slack in check.handlers`

For details about operators, read about the API response filtering operators.

Examples

Filter responses with label selectors

Use the `--label-selector` flag to filter responses using custom labels.

For example, to return entities with the `proxy_type` label set to `switch`:

```
sensuctl entity list --label-selector 'proxy_type == switch'
```

Filter responses with field selectors

Use the `--field-selector` flag to filter responses using specific resource attributes.

For example, to return entities with the `switches` subscription:

```
sensuctl entity list --field-selector 'switches in entity.subscriptions'
```

To retrieve all events that equal a status of `2` (CRITICAL):

```
sensuctl event list --field-selector 'event.check.status == "2"'
```

To retrieve all entities whose name includes the substring `webserver`:

```
sensuctl entity list --field-selector 'entity.name matches "webserver"'
```

Use the logical AND operator

To use the logical AND operator (`&&`) to return checks that include a `linux` subscription in the `dev` namespace:

```
sensuctl check list --field-selector 'linux in check.subscriptions && dev in
check.namespace'
```

Combine label and field selectors

You can combine the `--label-selector` and `--field-selector` flags in a single command.

For example, this command returns checks with the `region` label set to `us-west-1` that also use the `slack` handler:

```
sensuctl check list --label-selector 'region == "us-west-1"' --field-selector 'slack in check.handlers'
```

Set environment variables with sensuctl

Sensu allows you to set sensuctl environment variables for a single sensuctl command or with sensuctl configure. You can also export and set environment variables on your system with sensuctl env.

These environment variables are alternatives to configuration flags like the sensuctl global flags and sensuctl configure flags.

Setting sensuctl options as environment variables instead of using flags offers the following advantages:

- Use environment variables to avoid exposing sensitive information like your API key and other security credentials. Sensitive information is visible when you use command-line configuration flags.
- Inject exported environment variables for sensuctl commands in an automation script, such as a container creation script.
- Configure different shells for individual Sensu instances with the desired sets of environment variables rather than running sensuctl configure every time you want to switch between instances.

Set environment variables for a single command

Set certain environment variables for a single sensuctl command to temporarily override your current settings.

For example, to quickly check the entities in the `production` namespace while you are currently in the `default` namespace, run:

```
SENSU_NAMESPACE=production sensuctl entity list
```

Single-command environment variables are not persistent. To continue the example, if you run `sensuctl entity list` again, the response will include entities for the `default` namespace (not `production`).

These are the environment variables you can set for a single sensuctl command:

```
SENSU_API_KEY                    API key to use for authentication
SENSU_API_URL                    host URL of Sensu installation
SENSU_CACHE_DIR                  path to directory containing cache & temporary
files
SENSU_CONFIG_DIR                 path to directory containing configuration files
SENSU_INSECURE_SKIP_TLS_VERIFY   skip TLS certificate verification (Boolean value)
SENSU_NAMESPACE                  namespace in which to perform actions (default
"default")
SENSU_TIMEOUT                    timeout when communicating with sensu backend
(default 15s)
SENSU_TRUSTED_CA_FILE            TLS CA certificate bundle in PEM format
```

Set environment variables with sensuctl configure

To set certain environment variables with `sensuctl configure`, define the environment variables in the same command.For example:

```
SENSU_OIDC=true SENSU_NON_INTERACTIVE=true SENSU_FORMAT=yaml SENSU_PORT=7999
SENSU_TIMEOUT=49s SENSU_URL=http://192.168.7.217:8080 sensuctl configure
```

Environment variables set with `sensuctl configure` **are** persistent.

These are the environment variables you can set for sensuctl configure:

```
SENSU_FORMAT                     preferred output format (default "tabular")
SENSU_NON_INTERACTIVE            do not administer interactive questionnaire
SENSU_OIDC                       use an OIDC provider for authentication (Boolean
value)
SENSU_PASSWORD                   password
SENSU_PORT (used with SENSU_OIDC) port for local HTTP web server used for OAuth 2
callback during OIDC authentication (default 8000)
SENSU_URL                        the sensu backend url (default
"http://localhost:8080")
SENSU_USERNAME                   username
```

Export environment variables with sensuctl env

Export your shell environment with sensuctl env to use the exported environment variables with cURL and other scripts.

This example shows how to use sensuctl env to export environment variables and configure your shell:

BASH

```
export SENSU_API_URL="http://127.0.0.1:8080"
export SENSU_NAMESPACE="default"
export SENSU_FORMAT="tabular"
export SENSU_ACCESS_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
export SENSU_ACCESS_TOKEN_EXPIRES_AT="1567716187"
export SENSU_REFRESH_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
export SENSU_TRUSTED_CA_FILE=""
export SENSU_INSECURE_SKIP_TLS_VERIFY="true"
eval $(sensuctl env)
```

CMD

```
SET SENSU_API_URL=http://127.0.0.1:8080
SET SENSU_NAMESPACE=default
SET SENSU_FORMAT=tabular
SET SENSU_ACCESS_TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x
SET SENSU_ACCESS_TOKEN_EXPIRES_AT=1567716676
SET SENSU_REFRESH_TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x
SET SENSU_TRUSTED_CA_FILE=
SET SENSU_INSECURE_SKIP_TLS_VERIFY=true
@FOR /f "tokens=*" %i IN ('sensuctl env --shell cmd') DO @%i
```

POWERSHELL

```
$Env:SENSU_API_URL = "http://127.0.0.1:8080"
$Env:SENSU_NAMESPACE = "default"
$Env:SENSU_FORMAT = "tabular"
$Env:SENSU_ACCESS_TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
```

```
$Env:SENSU_ACCESS_TOKEN_EXPIRES_AT = "1567716738"
$Env:SENSU_REFRESH_TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
$Env:SENSU_TRUSTED_CA_FILE = ""
$Env:SENSU_INSECURE_SKIP_TLS_VERIFY = "true"
& sensuctl env --shell powershell | Invoke-Expression
```

***NOTE**: If you receive an `invalid credentials` error while using sensuctl env, run `eval $(sensuctl env)` to refresh your token.*

The `sensuctl env` command allows you to export the following environment variables:

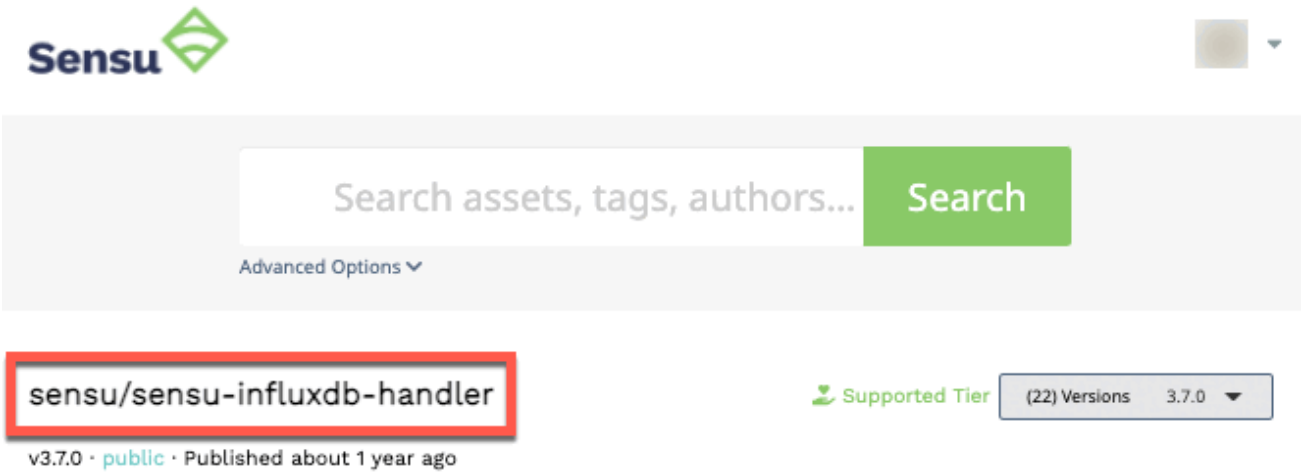
```
SENSU_API_KEY                   API key to use for authentication
SENSU_API_URL                   URL of the Sensu backend API in sensuctl
SENSU_NAMESPACE                 Name of the current namespace in sensuctl
SENSU_FORMAT                    Set output format in sensuctl (for example, JSON,
YAML, etc.)
SENSU_ACCESS_TOKEN              Current API access token in sensuctl
SENSU_ACCESS_TOKEN_EXPIRES_AT   Timestamp specifying when the current API access
token expires
SENSU_REFRESH_TOKEN             Refresh token used to obtain a new access token
SENSU_TIMEOUT                   timeout when communicating with sensu backend
(default 15s)
SENSU_TRUSTED_CA_FILE           Path to a trusted CA file if set in sensuctl
SENSU_INSECURE_SKIP_TLS_VERIFY  Boolean value that can be set to skip TLS
verification
```

Use sensuctl with Bonsai

Sensuctl supports installing dynamic runtime asset definitions directly from Bonsai, the Sensu asset hub, and checking your Sensu backend for outdated dynamic runtime assets. You can also use `sensuctl command` to install, execute, list, and delete commands from Bonsai or a URL.

Install dynamic runtime asset definitions

To install a dynamic runtime asset definition directly from Bonsai, use `sensuctl asset add <ASSET_NAME>:<ASSET_VERSION>`. Replace `<ASSET_NAME>` with the complete name of the dynamic runtime asset from Bonsai. An asset's complete name includes both the part before the forward slash (sometimes called the Bonsai namespace) and the part after the forward slash.



Replace `<ASSET_VERSION>` with the asset version you want to install. To automatically install the latest version of the plugin, you do not need to specify the version: `sensuctl asset add <ASSET_NAME>`.

> ***NOTE**: Specify the asset version you want to install to maintain the stability of your observability infrastructure. If you do not specify a version to install, Sensu automatically installs the latest version, which may include breaking changes.*

For example, to install version 3.7.0 of the sensu/sensu-influxdb-handler dynamic runtime asset:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.7.0
```

The response should be similar to this example:

```
fetching bonsai asset: sensu/sensu-influxdb-handler:3.7.0
added asset: sensu/sensu-influxdb-handler:3.7.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu/sensu-influxdb-handler"].
```

You can also use the `--rename` flag to rename the dynamic runtime asset on install:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.7.0 --rename influxdb-handler
```

***NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

Check your Sensu backend for outdated dynamic runtime assets

To check your Sensu backend for dynamic runtime assets that have newer versions available on Bonsai, use `sensuctl asset outdated`. This will print a list of dynamic runtime assets installed in the backend whose version is older than the newest version available on Bonsai:

```
sensuctl asset outdated
```

If outdated assets are installed on the backend, the response will be similar to this example:

```
          Asset Name                   Bonsai Asset          Current Version  Latest
Version
---------------------------  ---------------------------  ---------------  -------
-------
sensu/sensu-influxdb-handler  sensu/sensu-influxdb-handler       3.6.1
3.7.0
```

Extend sensuctl with commands

Use `sensuctl command` to install, execute, list, and delete commands from Bonsai or a URL.

Install commands

To install a sensuctl command from Bonsai or a URL:

```
sensuctl command install <ALIAS> (<ASSET_NAME>:<ASSET_VERSION> | --url <ARCHIVE_URL>
--checksum <ARCHIVE_CHECKSUM>) <FLAGS>
```

To install a command plugin, use the Bonsai asset name or specify a URL and SHA512 checksum.

To install a command using the Bonsai asset name, replace `<ASSET_NAME>` with the complete name of the asset from Bonsai. `:<ASSET_VERSION>` is only required if you require a specific version or are pinning to a specific version.If you do not specify a version, sensuctl will fetch the latest version from Bonsai.

Replace `<ALIAS>` with a unique name for the command.For example, for the Sensu EC2 Discovery Plugin, you might use the alias `sensu-ec2-discovery`. `<ALIAS>` is required.

Replace `<FLAGS>` with the flags you want to use.Run `sensuctl command install -h` to view flags. Flags are optional and apply only to the `install` command — they are not saved as part of the command you are installing.

To install a command from the Sensu EC2 Discovery Plugin with no flags:

```
sensuctl command install sensu-ec2-discovery portertech/sensu-ec2-discovery:0.3.0
```

To install a command from a URL, replace `<ARCHIVE_URL>` with a command URL that points to a tarball (for example, https://path/to/asset.tar.gz).Replace `<ARCHIVE_CHECKSUM>` with the checksum you want to use.Replace `<ALIAS>` with a unique name for the command.

Replace `<FLAGS>` with the flags you want to use.Run `sensuctl command install -h` to view flags. Flags are optional and apply only to the `install` command — they are not saved as part of the command you are installing.

For example, to install a command-test dynamic runtime asset via URL with no flags:

```
sensuctl command install command-test --url https://github.com/amdprophet/command-test/releases/download/v0.0.4/command-test_0.0.4_darwin_amd64.tar.gz --checksum 8b15a170e091dab42256fe64ca7c4a050ed49a9dbfd6c8129c95506a8a9a91f2762ac1a6d24f4fc545430613fd45abc91d3e5d3605fcfffb270dcf01996caa7f
```

***NOTE**: Dynamic runtime asset definitions with multiple asset builds are only supported via Bonsai.*

Execute commands

To execute a sensuctl command plugin via its dynamic runtime asset's bin/entrypoint executable:

```
sensuctl command exec <ALIAS> <GLOBAL_FLAGS> <FLAGS>
```

Replace `<ALIAS>` with a unique name for the command.For example, for the Sensu EC2 Discovery Plugin, you might use the alias `sensu-ec2-discovery`.`<ALIAS>` is required.

Replace `<GLOBAL_FLAGS>` with the globlal flags you want to use.Run `sensuctl command exec -h` to view global flags.To pass `<GLOBAL_FLAGS>` flags to the bin/entrypoint executable, make sure to specify them after a double dash surrounded by spaces.

Replace `<FLAGS>` with the flags you want to use.Run `sensuctl command exec -h` to view flags. Flags are optional and apply only to the `exec` command — they are not saved as part of the command you are executing.

***NOTE**: When you use `sensuctl command exec`, the environment variables are passed to the command.*

For example:

```
sensuctl command exec <COMMAND> <GLOBAL_FLAG_1> <GLOBAL_FLAG_2> --cache-dir /tmp ---<FLAG_1> --<FLAG_2>=<value>
```

Sensuctl will parse the –cache-dir flag, but bin/entrypoint will parse all flags after the `--`.

In this example, the full command run by sensuctl exec would be:

```
bin/entrypoint <GLOBAL_FLAG_1> <GLOBAL_FLAG_2> --<FLAG_1> --<FLAG_2>=<value>
```

List commands

To list installed sensuctl commands:

```
sensuctl command list <FLAGS>
```

Replace `<FLAGS>` with the flags you want to use.Run `sensuctl command list -h` to view flags. Flags are optional and apply only to the `list` command.

Delete commands

To delete sensuctl commands:

```
sensuctl command delete <ALIAS> <FLAGS>
```

Replace `<ALIAS>` with a unique name for the command.For example, for the sensu/sensu-ec2-handler, you might use the alias `sensu-ec2-handler`.`<ALIAS>` is required.

Replace `<FLAGS>` with the flags you want to use.Run `sensuctl command delete -h` to view flags. Flags are optional and apply only to the `delete` command.

Web UI

***COMMERCIAL FEATURE**: Access the Sensu web UI in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu backend includes the **Sensu web UI**: a unified view of your events, entities, and checks with user-friendly tools that provide single-pane-of-glass visibility and reduce alert fatigue.

The web UI homepage provides a high-level overview of the overall health of the systems under Sensu's management, with a summary of active incidents, the number of incidents by severity, the types of entities under management, and the numbers of entities and incidents per namespace.

Access the web UI

After you start the Sensu backend, you can access the web UI in your browser by visiting http://localhost:3000.

***NOTE**: You may need to replace `localhost` with the hostname or IP address where the Sensu backend is running.*

Sign in to the web UI

Sign in to the web UI with the username and password you used to configure sensuctl.

The web UI uses your username and password to obtain access and refresh tokens via the Sensu /auth API. The access and refresh tokens are JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature. The access and refresh tokens are saved in your browser's local storage.

The web UI complies with Sensu role-based access control (RBAC), so individual users can view information according to their access configurations. Read the RBAC reference for default user credentials and instructions for creating new users.

View system information

Press `CTRL .` in the web UI to open the system information modal and view information about your Sensu backend and etcd or PostgreSQL datastore. For users with permission to create or update licenses, the system information modal includes license expiration information.

License expiration banner

A banner appears at the top of the web UI screen when your organization's license is expiring:

The banner is only visible to users who have read access to your organization's license.

By default, the banner starts appearing when the license expiration is 30 days away. To adjust the number of days before license expiration to begin displaying the banner, use the license_expiry_reminder web UI configuration attribute.

Use the implicit OR operator

On the Sensu web UI homepage, you can use the search function to limit the display by cluster and namespace. If you specify the same attribute twice with different values, Sensu automatically applies a logical OR operator to your search.

For example, suppose you enter two search expressions in the search bar on the web UI homepage: `namespace: devel_1` and `namespace: devel_2`. In this case, the web UI homepage will display all data for both namespaces: `devel_1` and `devel_2`.

Change web UI themes

Use the preferences menu to change the theme or switch to the dark theme.

Troubleshoot web UI errors

Read Troubleshoot Sensu to resolve and investigate web UI errors.

View and manage resources in the web UI

***COMMERCIAL FEATURE**: Access the web UI in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

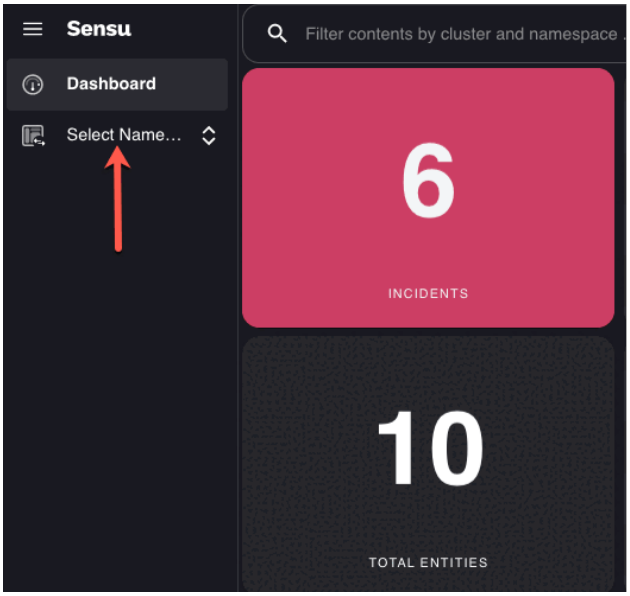
You can view and manage Sensu resources in the web UI, including events, entities, silences, checks, handlers, event filters, and mutators.

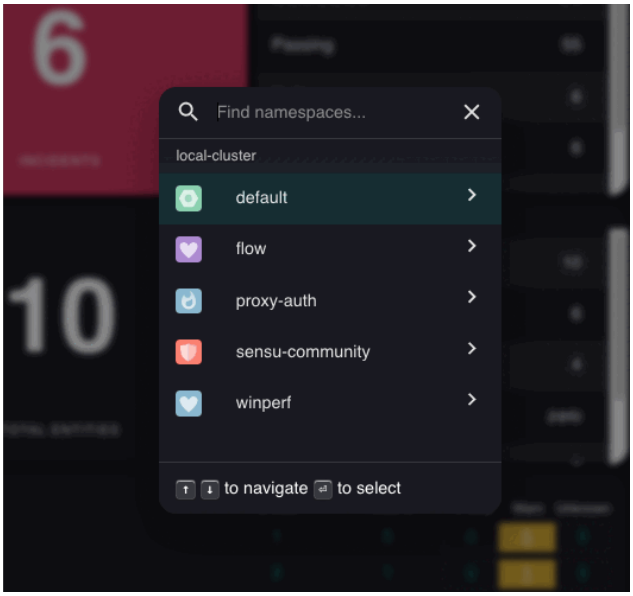
Use the namespace switcher

Beyond the homepage, the web UI displays events, entities, and resources for a single namespace at a time. By default, the web UI displays the `default` namespace.

To switch namespaces, select the menu icon in the upper-left corner or press the `Ctrl+K` keyboard shortcut and choose a namespace from the dropdown.

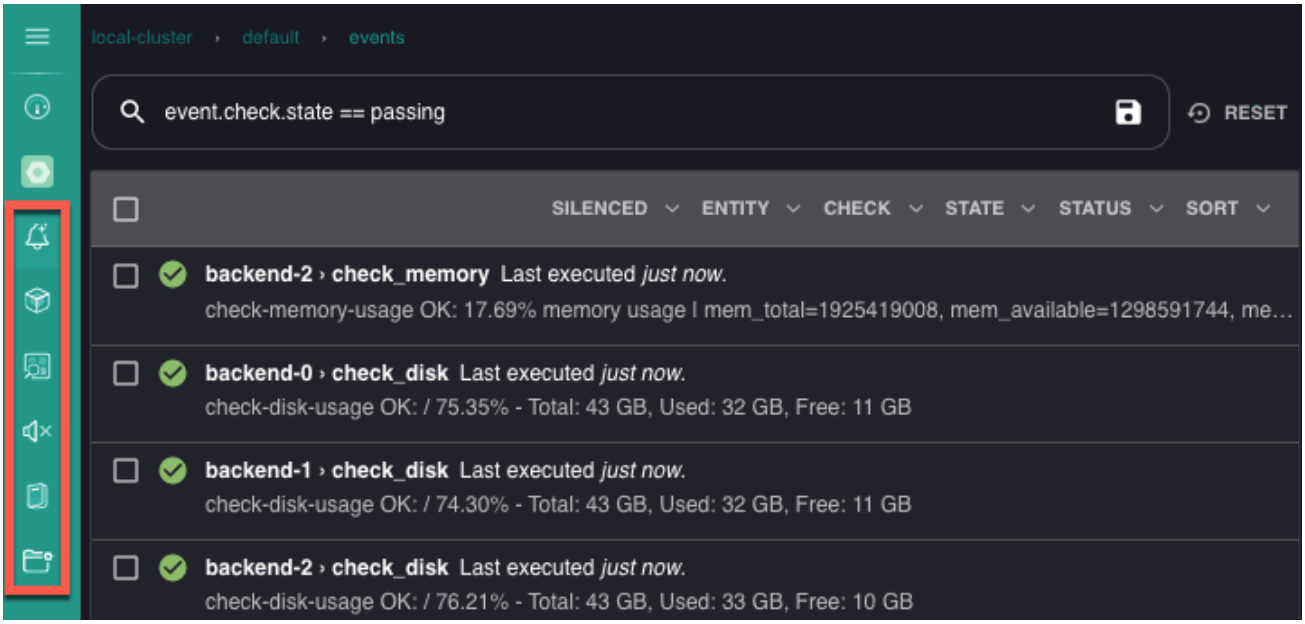
***NOTE**: The namespace switcher will list only the namespaces to which the current user has access.*



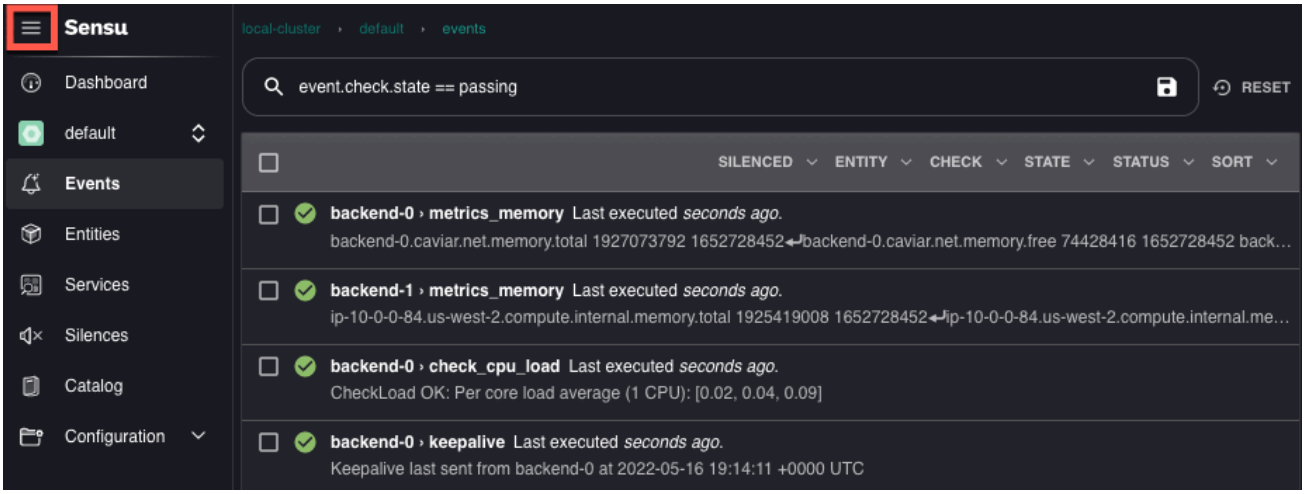


When you switch to a namespace, the left navigation menu loads so you can select specific pages for events, entities, services, silences, catalog, and configuration, which includes checks, handlers, event

filters, and mutators:

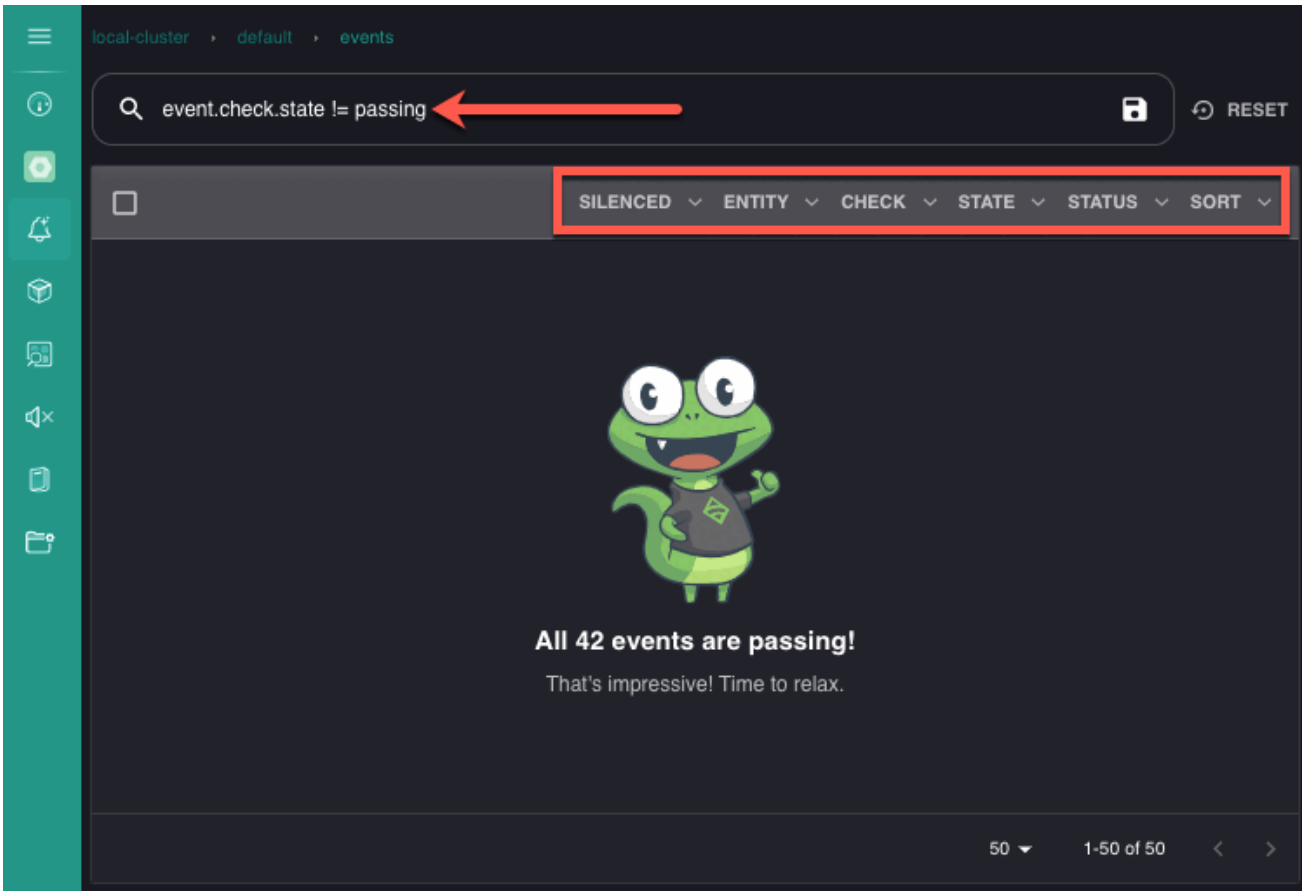


Click the ☰ icon at the top of the left-navigation menu to expand the menu and display page labels:

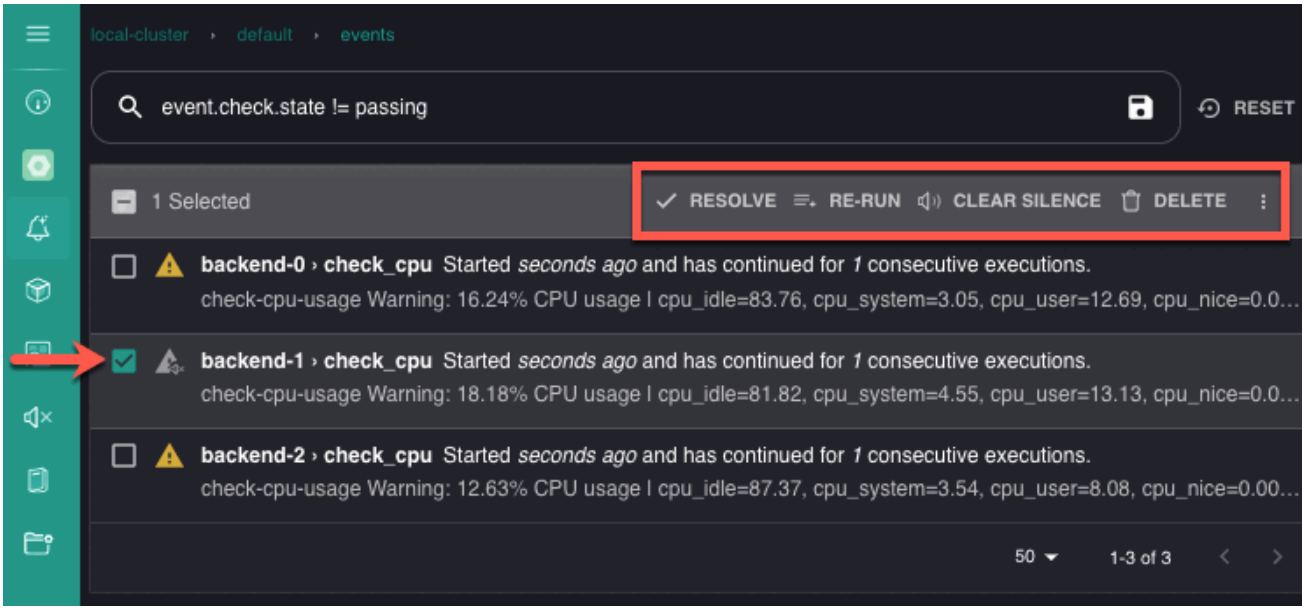


Manage events

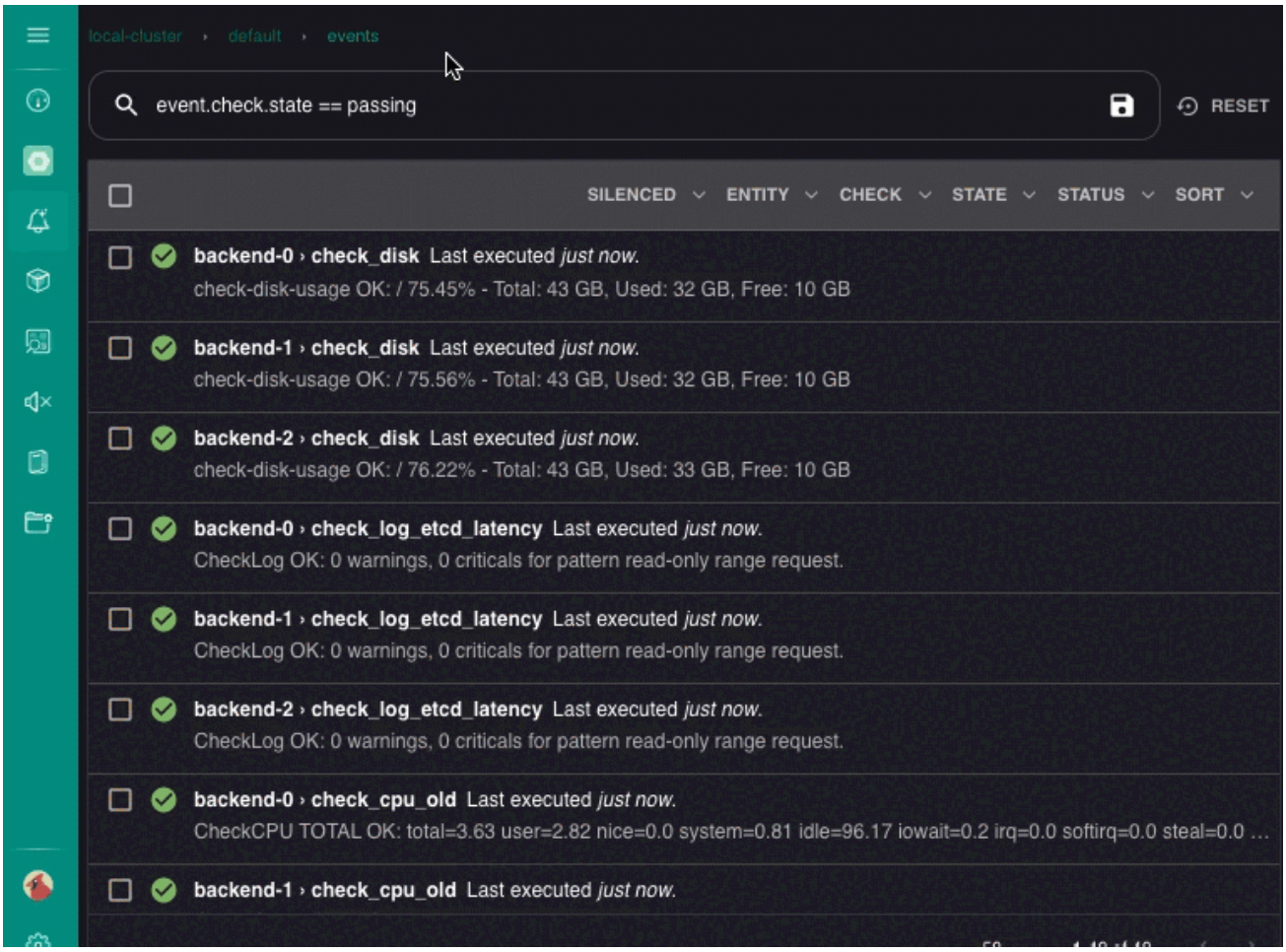
The Events page opens by default when you navigate to a namespace, with an automatic filter to show only events with a non-passing status (i.e. `event.check.state != passing`).The top row of the events list includes several other options for filtering and sorting events:



Click the check boxes to select one or more events and resolve, silence, or delete them directly from the Events page:

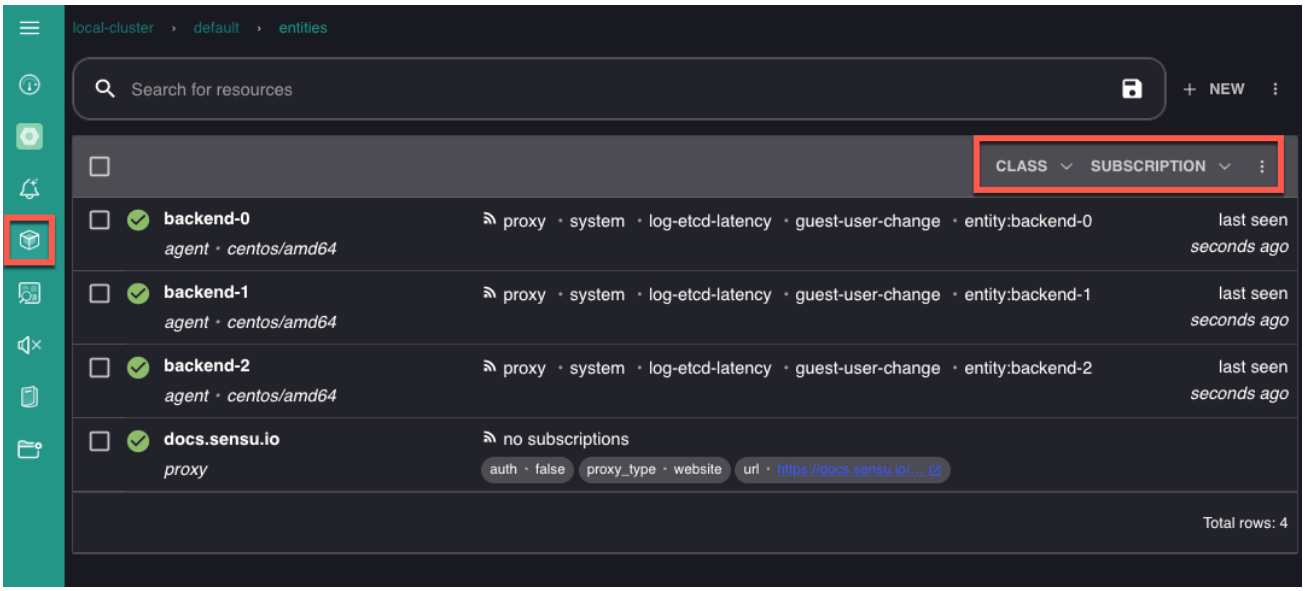


Click an event name to view details like status, output, number of occurrences, labels and annotations, related check configuration (if the event was produced by a service check), and entity summary, as well as a timeline that displays the event's last 20 statuses at a glance:

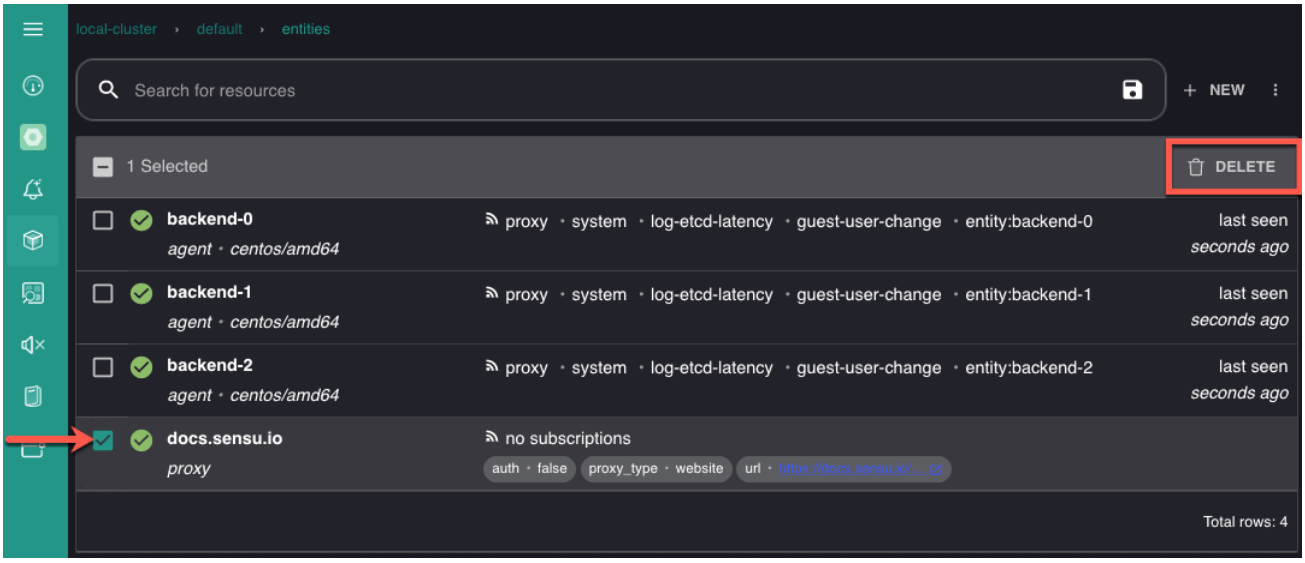


Manage entities

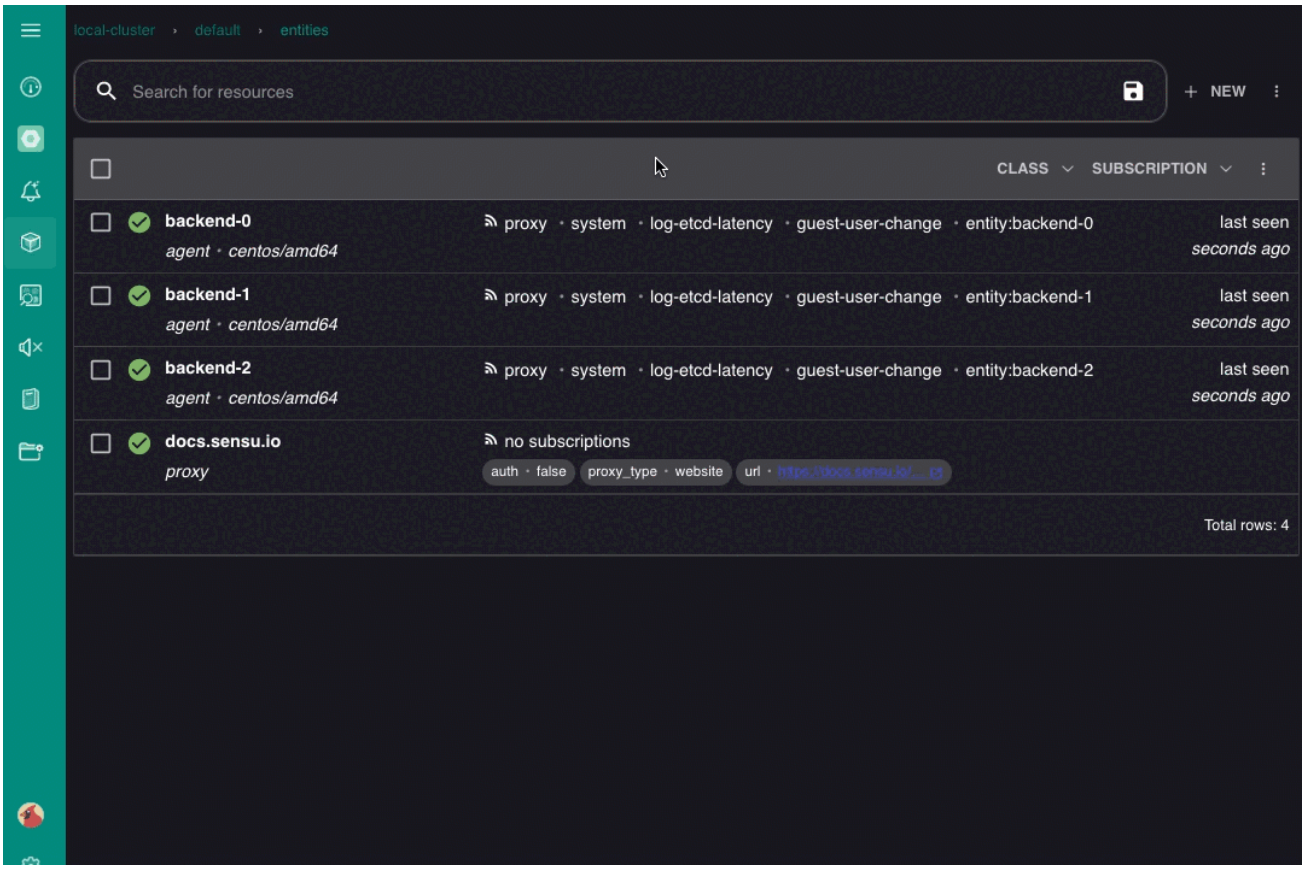
The Entities page provides real-time inventory information for the namespace's endpoints under Sensu management.The top row of the entities list includes options for filtering and sorting entities on the page:



Click the check boxes to select one or more entities and delete them directly from the Entities page:



Click an entity name to view details about the entity's creator, agent version (for agent entities), subscriptions, labels and annotations, associated events, and properties:

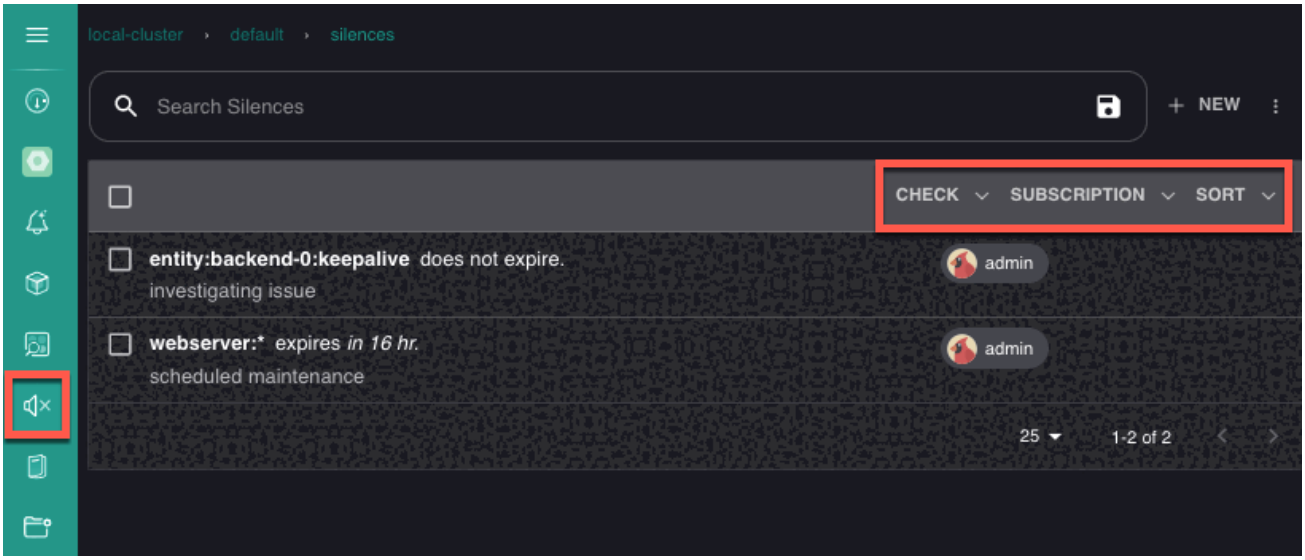


Manage services

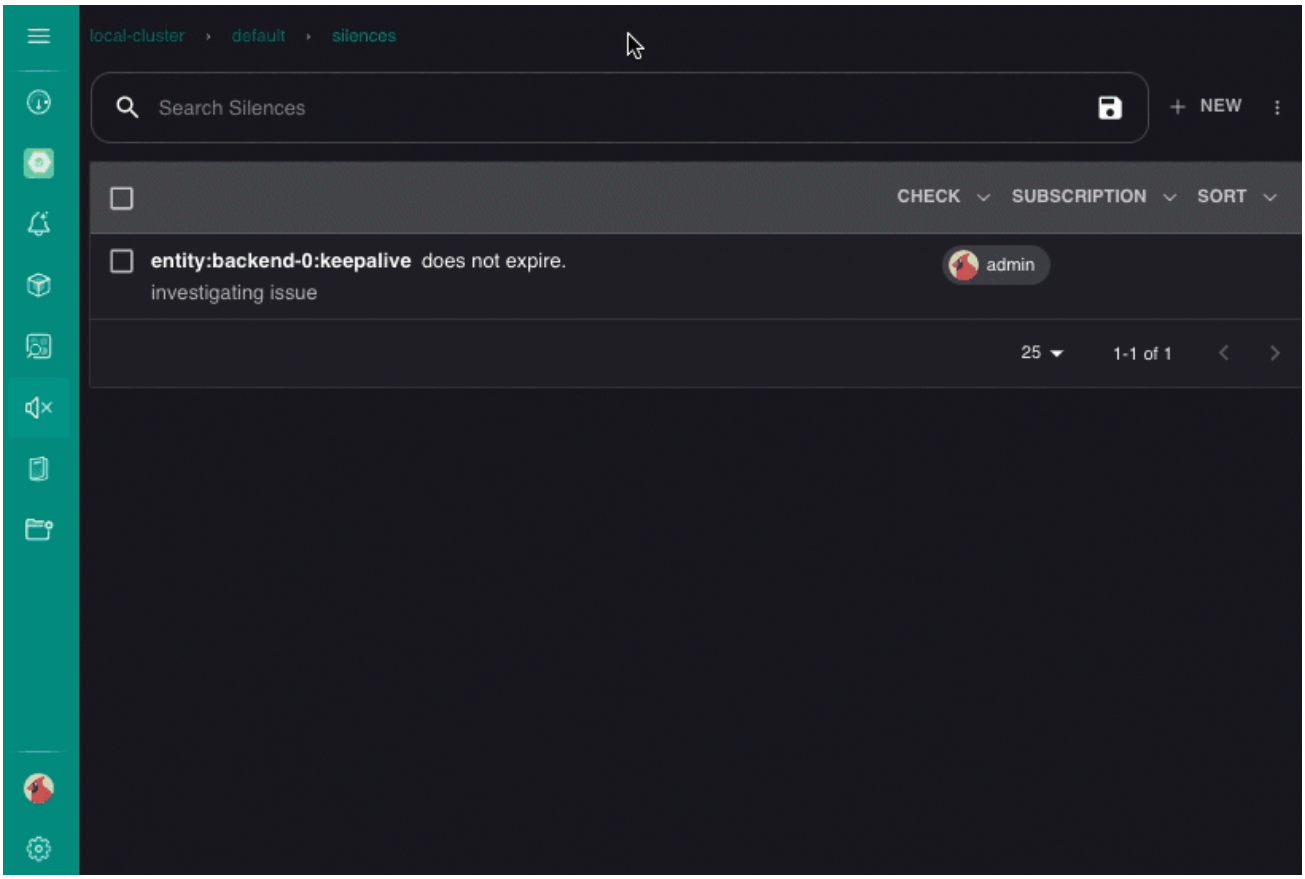
The Services page includes a module to help you build and configure service entities with service components and rule templates for business service monitoring (BSM).Read Build business service monitoring for details about the web UI BSM module.

Manage silences

Create silences by check or subscription name and clear silences in the web UI Silences page.The Silences page lists all active silences for the namespace.The top row of the silences list includes options for filtering and sorting silences on the page:



Click `+ NEW` to open a dialog window and create silences for individual events, by check or subscription name, or by entity:

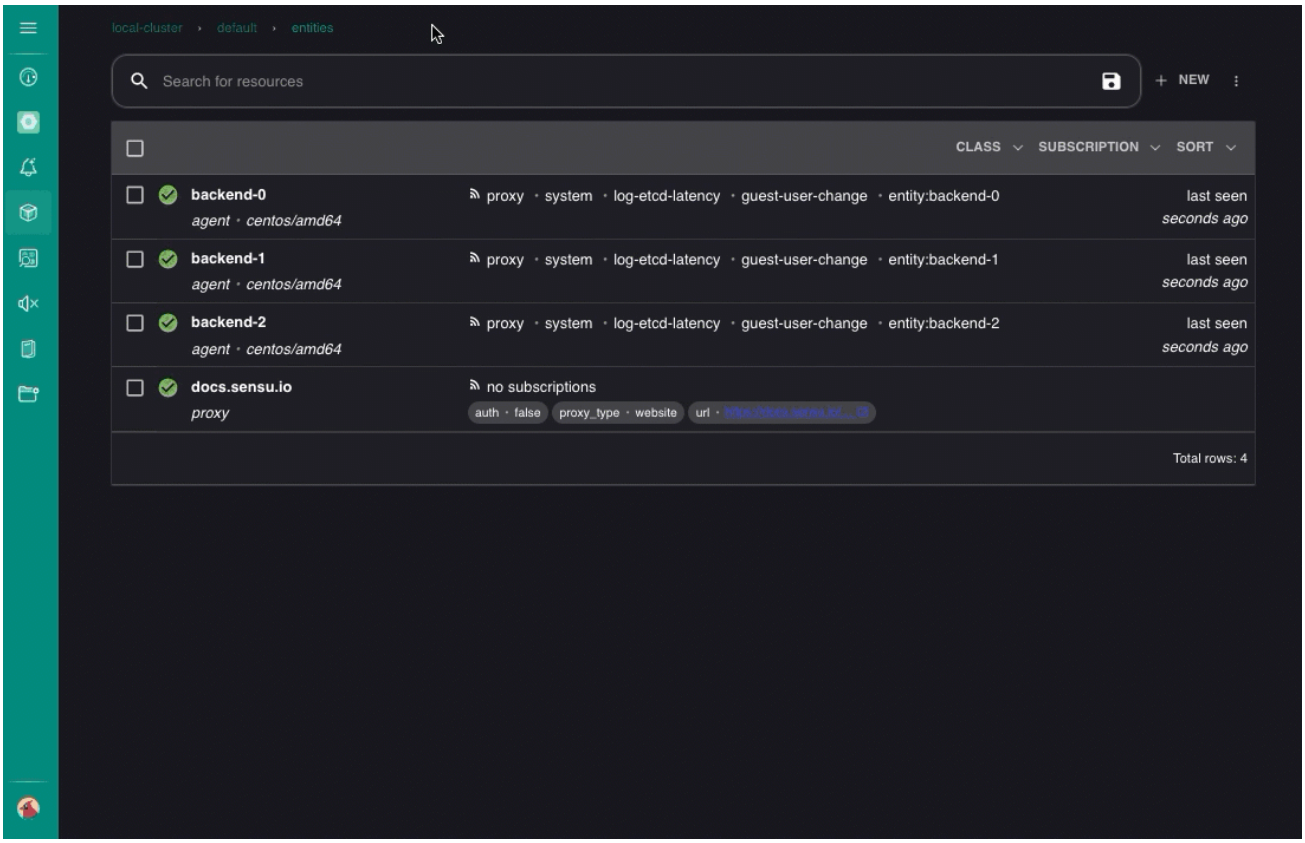


You can also silence individual checks and entities from their detail pages in the web UI.

After you create a silence, it will be listed in the web UI Silences page until you clear the silence or the silence expires.

Manage configuration resources

Under the Configuration menu option, you can access assets, checks, event filters, handlers, mutators, pipelines, role-based access control (RBAC) resources, and secrets. Each resource page lists the namespace's resources. The top row of each page includes options for filtering and sorting the listed resources.



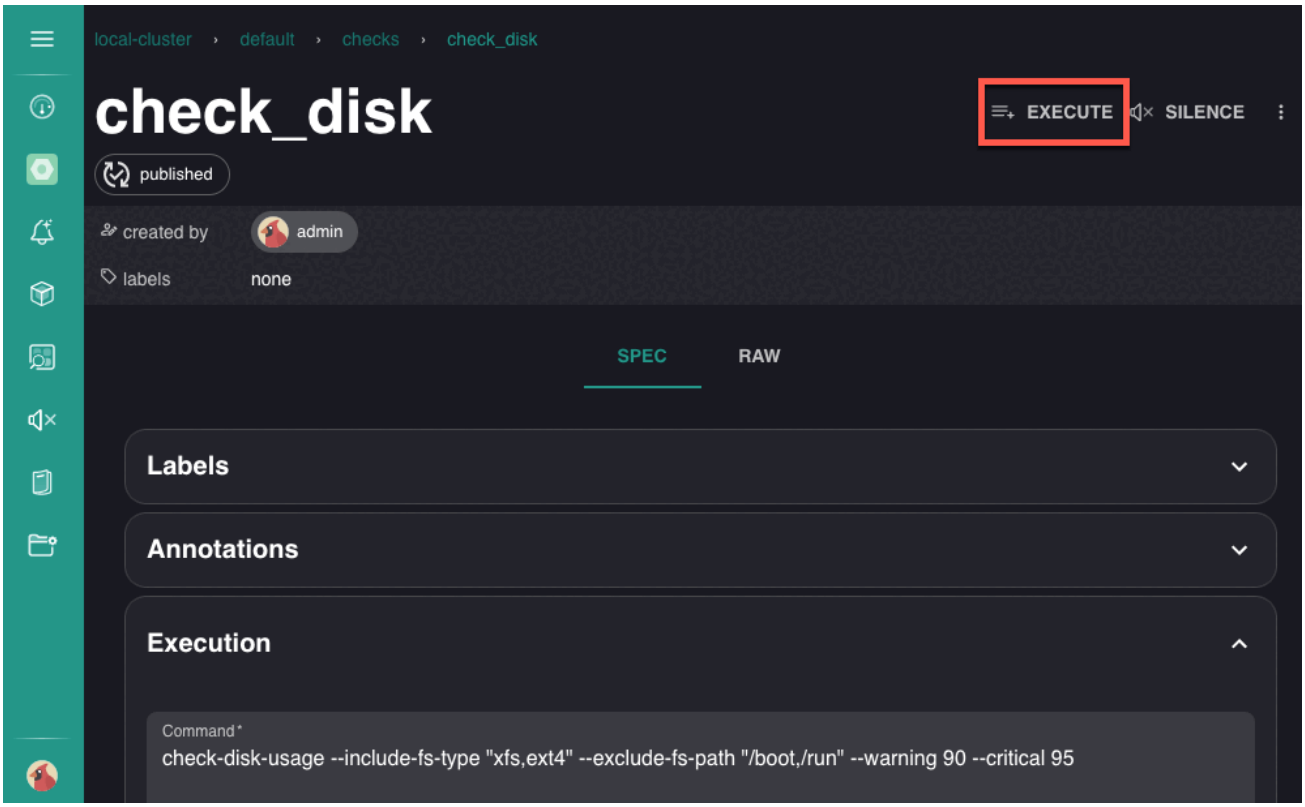
Click a resource name to view the resource's detail page, where you can review more information about the resource and edit or delete it.

On the Checks page, click the check boxes to select one or more checks to execute, silence, unpublish, or delete them.

Execute checks on demand

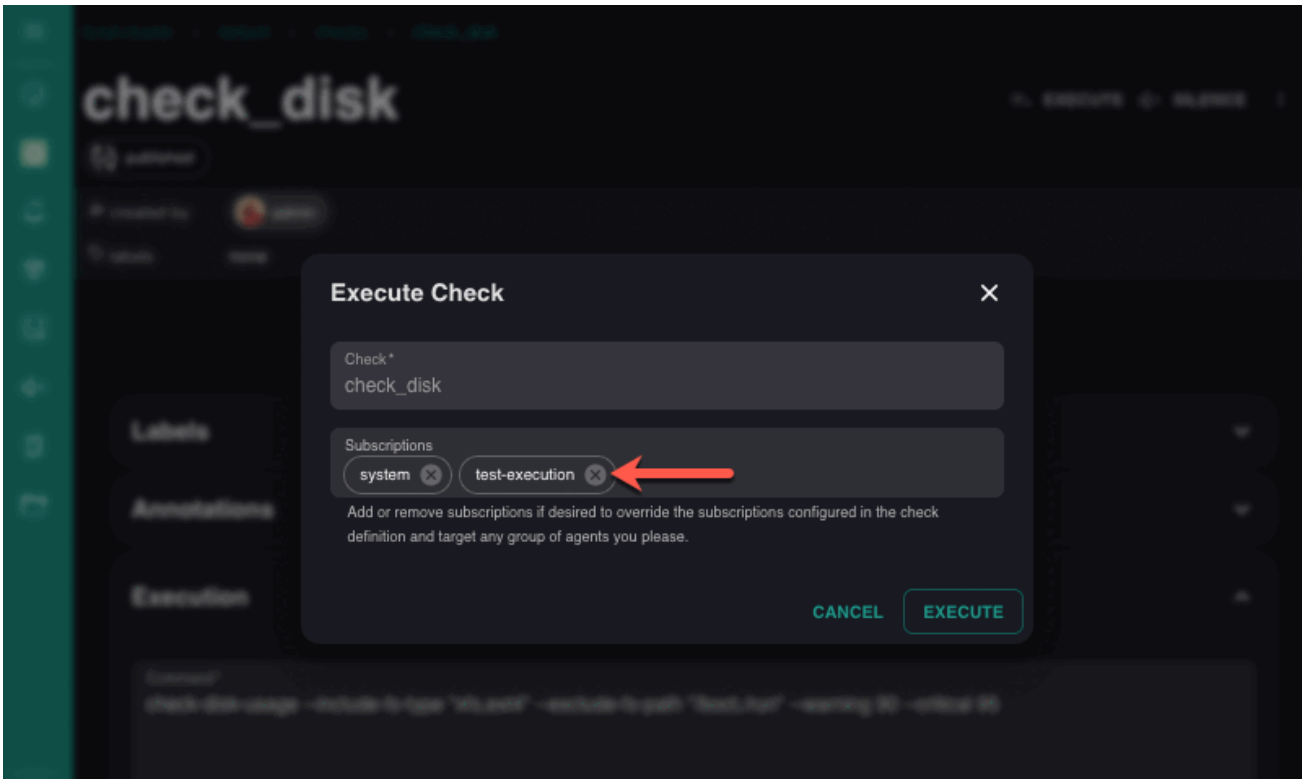
You can execute individual checks on demand and on any agent from each check's detail page to test your observability pipeline.

Click **EXECUTE** to open the Execute Check dialog window:



In the Execute Check dialog window, you can execute the check according to its existing subscriptions or add and remove subscriptions to execute it on specific agents.

> ***NOTE**: Changing the subscriptions for ad hoc execution in the Execute Check dialog window will not make any changes to the existing subscriptions in the check definition.*



***NOTE**: If you manually execute a round robin check in the web UI, Sensu will execute the check on all subscribed agents. After the manual execution, the check will run as scheduled in round robin fashion.*

To manually execute a round robin check on a single agent, specify the agent's entity name subscription in the Execute Check dialog. For example, if the entity is named `webserver1`, use the subscription `entity:webserver1`.

View resource data in the web UI

You can view and copy the YAML or JSON definition for any event, entity, or configuration resource directly in the web UI.

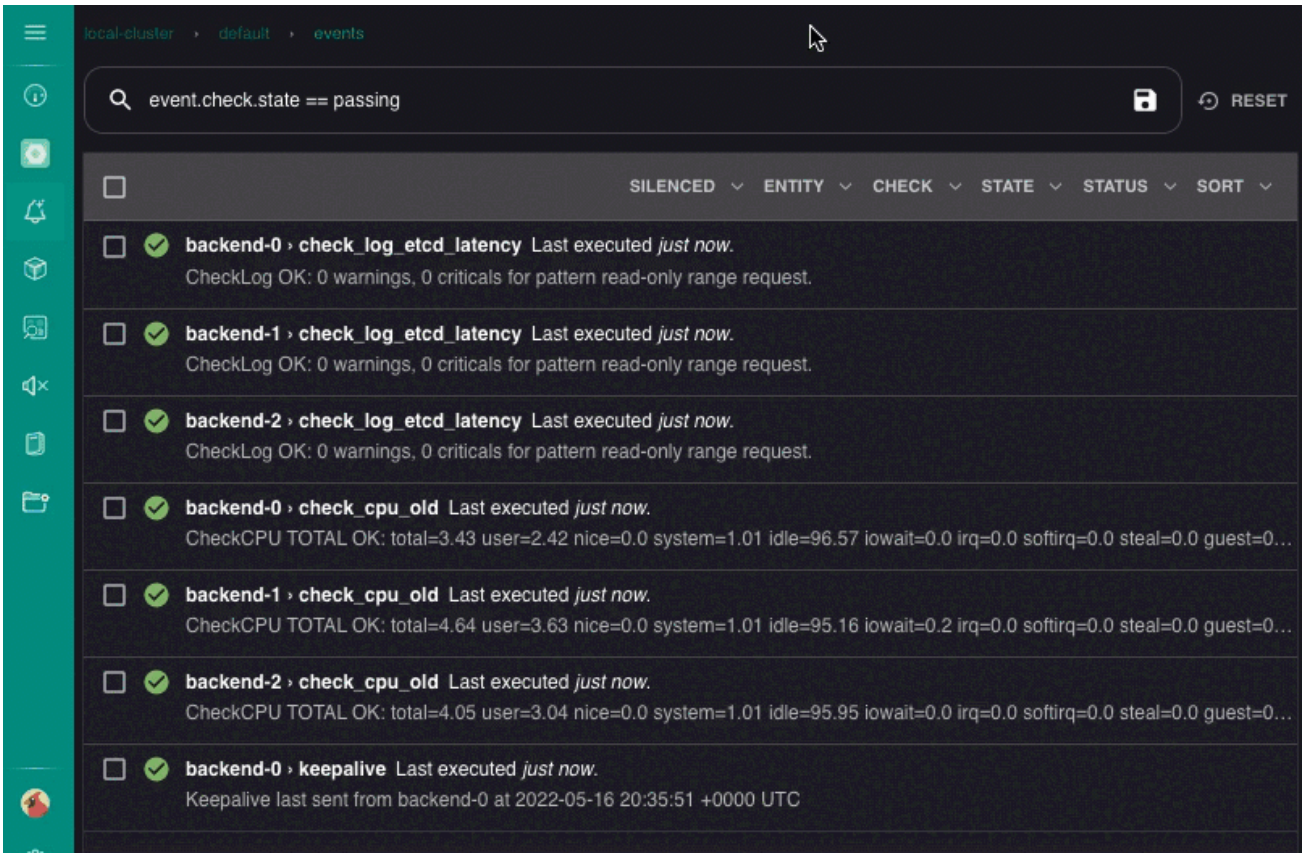
View resource data for an event or entity

To view and copy the YAML and JSON definitions for any event or entity in the web UI:

1. Open the individual resource page for the event or entity.
2. Click ⋮ at the top-right of the page.
3. Select **</> Data** to open the Resource Data dialog window.
4. In the Resource Data window, click the **yaml** or **json** button to select the format.

5. Click the copy button at the top-right of the Resource Data window to copy the resource definition.

This example shows how to view and copy the resource data for an event:

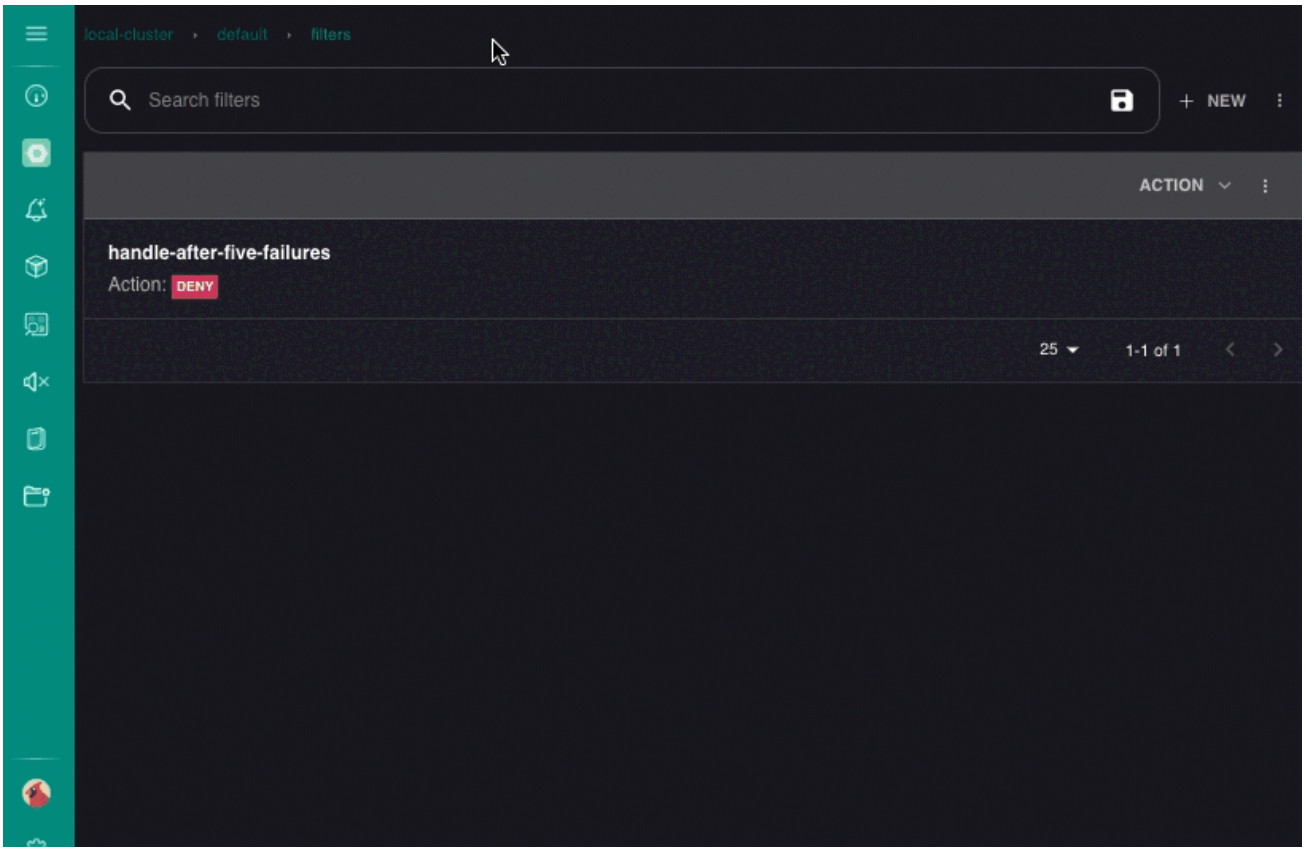


View resource data for a configuration resource

To view and copy the YAML and JSON definitions for any configuration resource in the web UI:

1. Open the individual resource page.
2. Click **RAW**.
3. In the resource data field, click the **yaml** or **json** button to select the format.
4. Click the copy button at the top-right of the resource data field to copy the resource definition.

This example shows how to view and copy the resource data for an event filter:


local-cluster
default
filters
Search filters
NEW
ACTION
handle-after-five-failures
Action:
DENY
25
1-1 of 1

Search in the web UI

***COMMERCIAL FEATURE**: Access the web UI, basic and advanced web UI searching, and saved searches in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu web UI includes basic search and filtering functions you can use to build customized views of your Sensu resources. Sensu also supports advanced web UI searches based on a wider range of resource attributes and custom labels as a commercial feature.

When you apply a search to a web UI page, it creates a unique link for the page of search results. You can bookmark these links and share your favorite search combinations. You can also save your favorite searches.

Events and entities search limits

If you use etcd for event storage, web UI search queries on the events and entities pages will stop after returning a certain number of matches. Without these limits, the search operation can diminish cluster health.

If a web UI search reaches the limit for the events or entities page, the results count at the bottom-right corner of the page will indicate that the total number of matches exceeds the number of results listed.

Events search limit

On the events page, if you use etcd for event storage, search queries will return a maximum of 50,000 events. For example, if you use etcd for event storage and you search in a namespace that has more than 50,000 matching events, the search results will not include matching events beyond the first 50,000.

Entities search limit

If you use etcd for event storage, search queries on the entities page will stop after retrieving approximately 500 matches. As a result, if your search matches more than 500 entities, the total results

count at the bottom-right corner of the entities page will not accurately reflect the number of matching entities.

Search operators

Web UI search supports two equality-based operators, two set-based operators, one substring matching operator, and one logical operator.

operator	description	example
`==`	Equality	`check.publish == "true"`
`!=`	Inequality	`check.namespace != "default"`
`in`	Included in	`"linux" in check.subscriptions`
`notin`	Not included in	`"slack" notin check.handlers`
`matches`	Substring matching	`check.name matches "linux-"`
`&&`	Logical AND	`check.publish == "true" && "slack" in check.handlers`

For details about operators, read about the API response filtering operators.

Use quick search

The web UI quick search allows you to query and filter Sensu resources without using search syntax. Type your search term into the search field on any page of the web UI and press `Enter`.Sensu will auto-complete a simple search statement for the resources on that page using substring matching.

For example, on the Events page in the web UI, if you type `mysql` into the search field, Sensu will auto-complete the search statement to `event.check.name matches "mysql"`.

Create basic searches

Sensu includes these basic search functions:

- **Events page**: search by entity, check, status, and silenced/unsilenced.
- **Entities page**: search by entity class and subscription.
- **Silences page**: search by check and subscription.
- **Checks page**: search by subscription and published/unpublished.
- **Handlers page**: search by handler type.
- **Filters page**: search by action.

If you are using the basic web UI search functions, you can create a search by clicking in the search bar at the top of the web UI page:

1. In the web UI, open the page of resources you want to search.
2. Click in the search bar at the top of the web UI page.
3. Select the attribute you want to search for from the dropdown list of options.
4. Click in the search bar again and select the search to apply.
5. Press **Return/Enter**.

> ***NOTE**: You do not need to specify a resource type in web UI search because you must navigate to the resource page before you construct the search.*

Create advanced searches

Sensu supports advanced web UI searches using a wider range of attributes, including custom labels. You can use the same methods, fields, and examples for web UI searches as for API response filtering, with some syntax differences.

To search resources based on fields and labels, you'll write a brief search statement.Depending on the operator you're using, the web UI search syntax is either:

```
<SEARCH_TERM> <OPERATOR> <FIELD>
```

or

```
```

```
<FIELD> <OPERATOR> <SEARCH_TERM>
```

Fields are specific resource attributes in dot notation. For example, this search will retrieve all events for entities with the `linux` subscription:

```
"linux" in event.entity.subscriptions
```

This search will retrieve all events that whose status is *not* equal to `passing`:

```
event.check.state != "passing"
```

To display only events for checks with the subscription `webserver`, enter this search statement on the **Events page**:

```
"webserver" in event.check.subscriptions
```

To display only checks that use the `slack` handler, enter this search statement on the **Checks page**:

```
"slack" in check.handlers
```

Search for numbers or special characters

If you are searching for a value that begins with a number, place the value in single or double quotes:

```
entity.name == '1b04994n'
entity.name == "1b04994n"
```

Likewise, to search string values that include special characters like hyphens and underscores, place the value in single or double quotes:

```
entity.labels.region == 'us-west-1'
entity.labels.region == "us-west-1"
```

To display only events at `2` (CRITICAL) status:

```
event.check.status == "2"
```

Search for labels

Labels are treated like any other field in web UI searches.

For example, to search based on a check label `version`, use:

```
check.labels.version matches "7"
```

To display only checks with the `type` label set to `server`, enter this search statement on the **Checks page**:

```
check.labels.type == "server"
```

To search for entities that are labeled for any region in the US (for example, `us-east-1`, `us-west-1`, and so on):

```
entity.labels.region matches "us"
```

***NOTE**: Web UI searches for label names that include hyphens are not supported.Searches that include a hyphenated label name, such as `entity.labels.imported-by`, will return an unsupported token error.*

Search for event labels

For label-based event searches, the web UI merges check and entity labels into a single search term: `event.labels.[KEY]`.

For example, to display events with the `type` label set to `server`, enter this search statement on the **Events** page:

```
event.labels.type == "server"
```

This search will retrieve events with the `type` label set to `server`, no matter whether the label is defined in the event's corresponding check or entity configuration.

Use the logical AND operator

To use the logical AND operator (`&&`) to return checks that include a `linux` subscription and the `slack` handler:

```
"linux" in check.subscriptions && "slack" in check.handlers
```

To return events that include a `windows` check subscription and any email handler:

```
"windows" in event.check.subscriptions && event.check.handlers matches "email"
```

Save a search

To save a web UI search:

1. Create a web UI search.
2. Click  at the right side of the search bar.
3. Click **Save this search**.
4. Type the name you want to use for the saved search.
5. Press **Return/Enter**.

Sensu saves your web UI searches to etcd in a namespaced resource named `searches`. To recall a saved web UI search, a Sensu user must be assigned to a role that includes permissions for both the `searches` resource and the namespace where you save the search.

The role-based access control (RBAC) reference includes example workflows that demonstrate how to configure a user's roles and role bindings to include full permissions for namespaced resources, including saved searches.

Recall a saved search

To recall a saved search, click  at the right side of the search bar and select the name of the search you want to recall.

You can combine an existing saved search with a new search to create a new saved search. To do this, recall a saved search, add the new search statement in the search bar, and save the combination as a new saved search.

Delete a saved search

To delete a saved search:

1. Click  at the right side of the search bar.
2. Find the saved search you want to delete and click the  next to it.

Use the sort function

Use the **SORT** dropdown menu to sort search results. You can sort all resources by name, but events and silences have additional sorting options:

- **Events page**: sort by last OK, severity, timestamp, and entity.
- **Silences page**: sort by start date.

Configure the web UI

***COMMERCIAL FEATURE**: Access web UI configuration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Web UI configuration allows you to define certain display options for the Sensu web UI, such as which web UI theme to use, the number of items to list on each page, and which URLs and linked images to expand. You can define a single custom web UI configuration to federate to all, some, or only one of your clusters.

Create a web UI configuration

Use the enterprise/web/v1 API POST endpoint or sensuctl create to create a `GlobalConfig` resource. The web UI configuration reference describes each attribute you can configure in the `GlobalConfig` resource.

***NOTE**: Each cluster should have only one web configuration.*

If an individual user's settings conflict with the web UI configuration settings, Sensu will use the individual user's settings. For example, if a user's system is set to dark mode and their web UI settings are configured to use their system settings, the web UI will use dark mode for that user, even if you set the theme to `classic` in your web UI configuration.

Federate a web UI configuration to specific clusters

The web UI configuration in use is provided by the cluster you are connected to. For example, if you open the web UI for https://cluster-a.sensu.my.org:3000, the web UI display will be configured according to the `GlobalConfig` resource for cluster-a.

In a federated environment, you can create an etcd replicator for your `GlobalConfig` resource so you can use it for different clusters:

YML

```
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: web_global_config
spec:
  api_version: web/v1
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  insecure: false
  key: /path/to/ssl/key.pem
  replication_interval_seconds: 120
  resource: GlobalConfig
  url: "http://127.0.0.1:2379"
```

JSON

```
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "web_global_config"
  },
  "spec": {
    "api_version": "web/v1",
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "insecure": false,
    "key": "/path/to/ssl/key.pem",
    "replication_interval_seconds": 120,
    "resource": "GlobalConfig",
    "url": "http://127.0.0.1:2379"
  }
}
```

Debugging in federated environments

In a federated environment, a problem like incorrect configuration, an error, or a network issue could

prevent a cluster from appearing in the web UI namespace switcher.

If you set the `always_show_local_cluster` attribute to `true` in your web UI configuration, the namespace switcher will display a heading for each federated cluster, along with the local-cluster heading to indicate the cluster you are currently connected to. With `always_show_local_cluster` set to `true`, the cluster administrator can directly connect to the local cluster even if there is a problem that would otherwise prevent the cluster from being listed in the namespace switcher.

> ***NOTE**: Use the `always_show_local_cluster` attribute only in federated environments. In a single-cluster environment, the namespace switcher will only list a local-cluster heading and the namespaces for that cluster.*

Build business service monitoring

***COMMERCIAL FEATURE**: Access the web UI and business service monitoring (BSM) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

The Sensu web UI includes a module to help you build and configure business service monitoring (BSM) service entities with service components and rule templates.

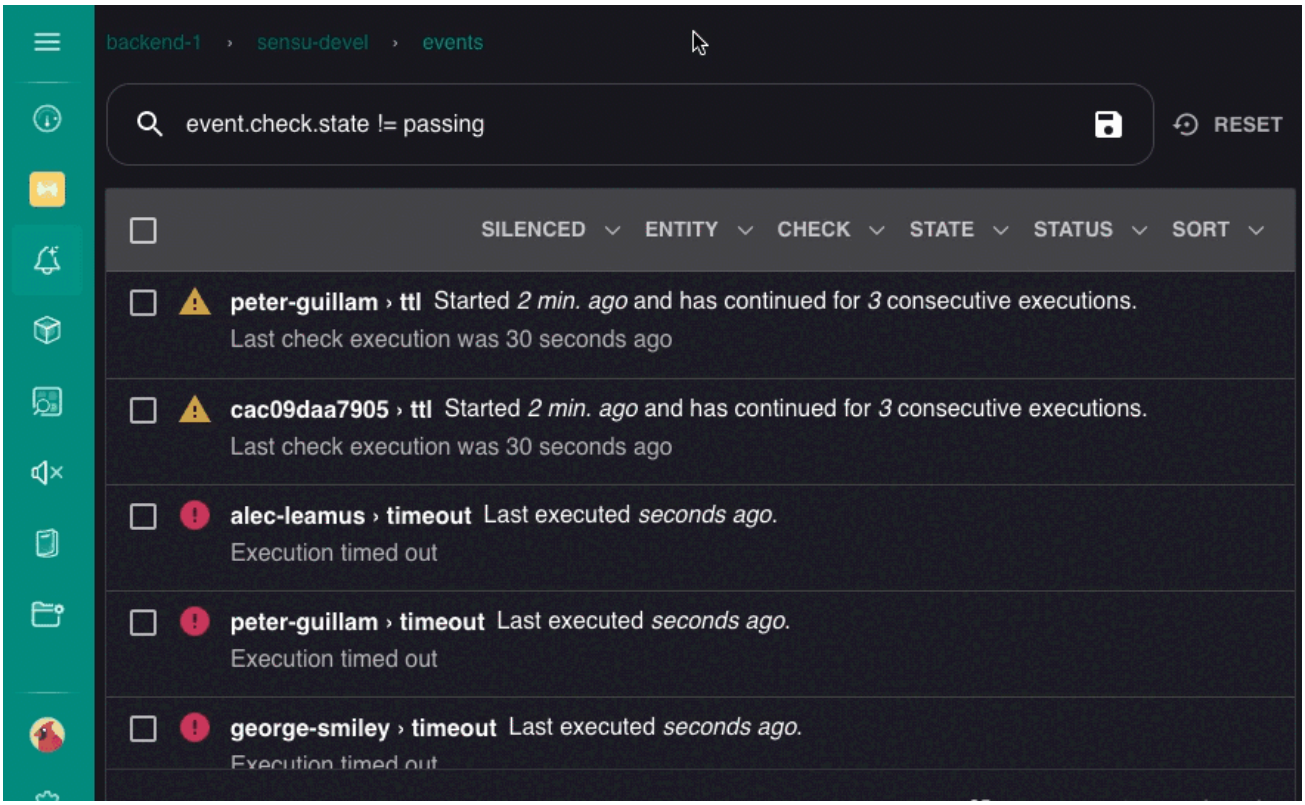
Build a business service

***NOTE**: BSM requires PostgreSQL to achieve high event throughput. For this reason, the web UI will display a PostgreSQL prompt instead of the BSM module until you configure a PostgreSQL datastore.*

To build a business service in the web UI module:

1. Click  in the left navigation menu to open the Services page.
2. Click **ADD NEW SERVICE** to open the Create New Service dialog window.
3. Enter a name for the service entity.
4. Enter labels and annotations, if desired.
5. Click **Submit**.

The updated Services page will include a tile for the new service:

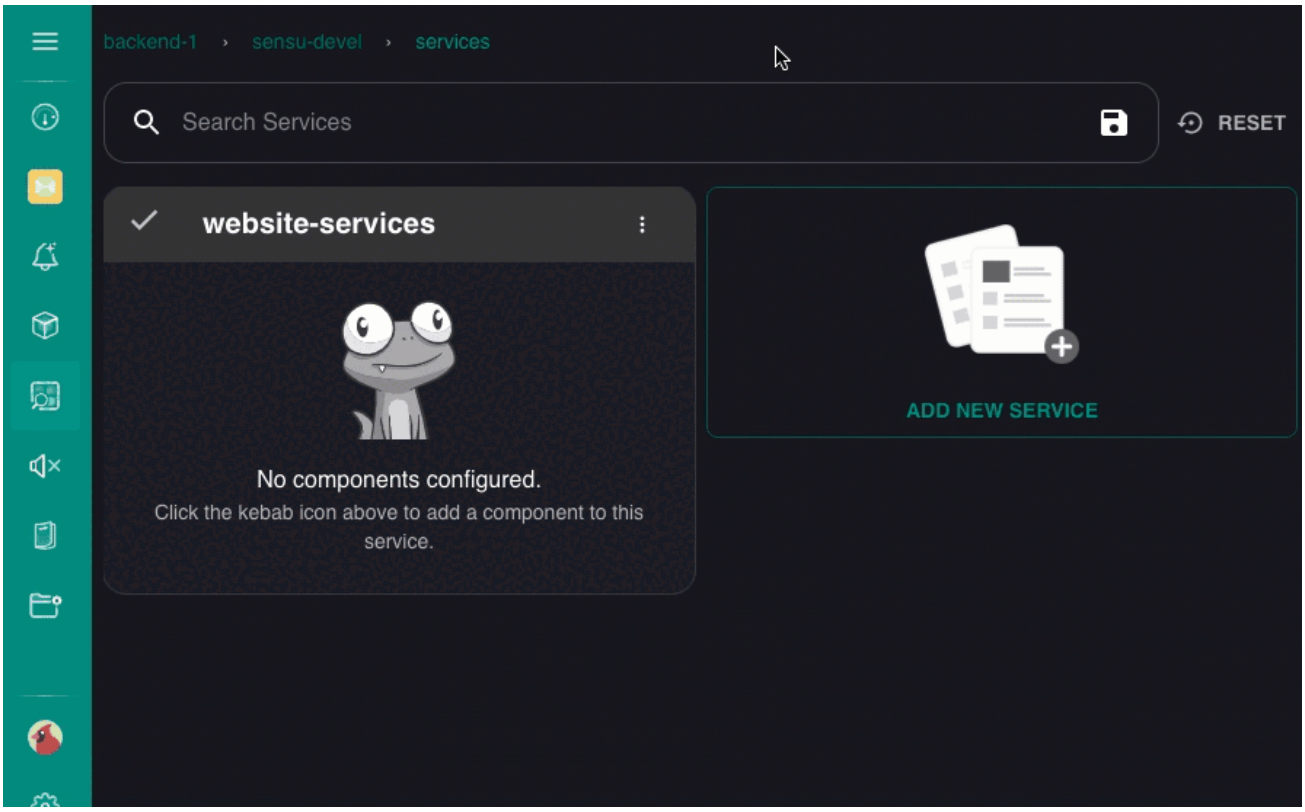


The business service itself is an entity with the class `service`, so it will also be listed on the Entities page.

To add service components to a business service:

1. Click ⋮ for the business service.
2. Select **+ New** from the drop-down menu to open the Configure New Service Component dialog window.
3. Enter a name for the service component.
4. Enter labels and annotations, if desired.
5. Enter query selectors to describe the events that each monitoring rule should process for the service component.
6. Select the rule template you wish to use and a unique name to use for the rule-specific events.
7. Enter values for the arguments to pass to the rule template. Available arguments will vary for different rule templates.
8. Specify the type of check scheduling the service component should use (interval or cron) as well as the desired interval in seconds or cron scheduling statement.
9. Specify the handlers the service component should use.
10. Click **Submit**.

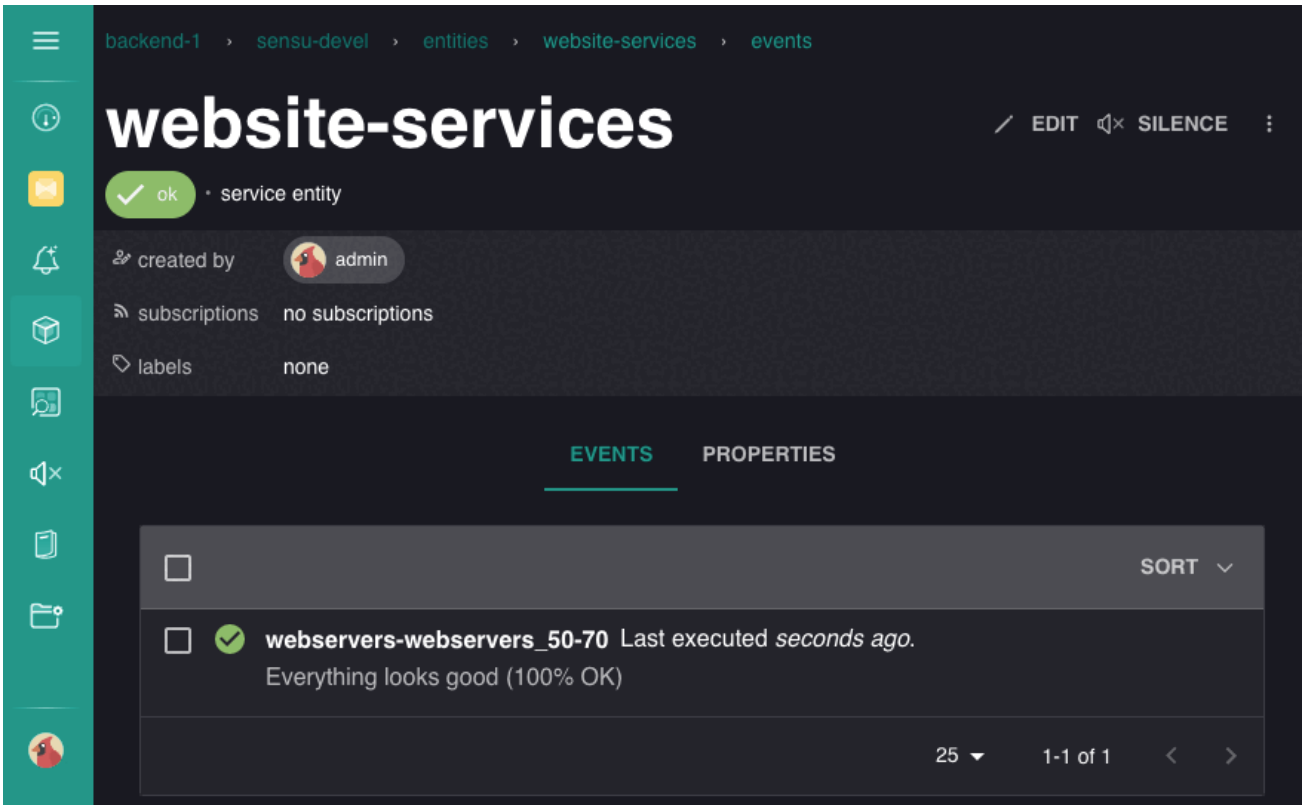
The updated business service tile will include the service component:



View and manage business services

After you create a business service by any means (web UI, API, or sensuctl), it will be listed in the web UI Services page until you delete it.

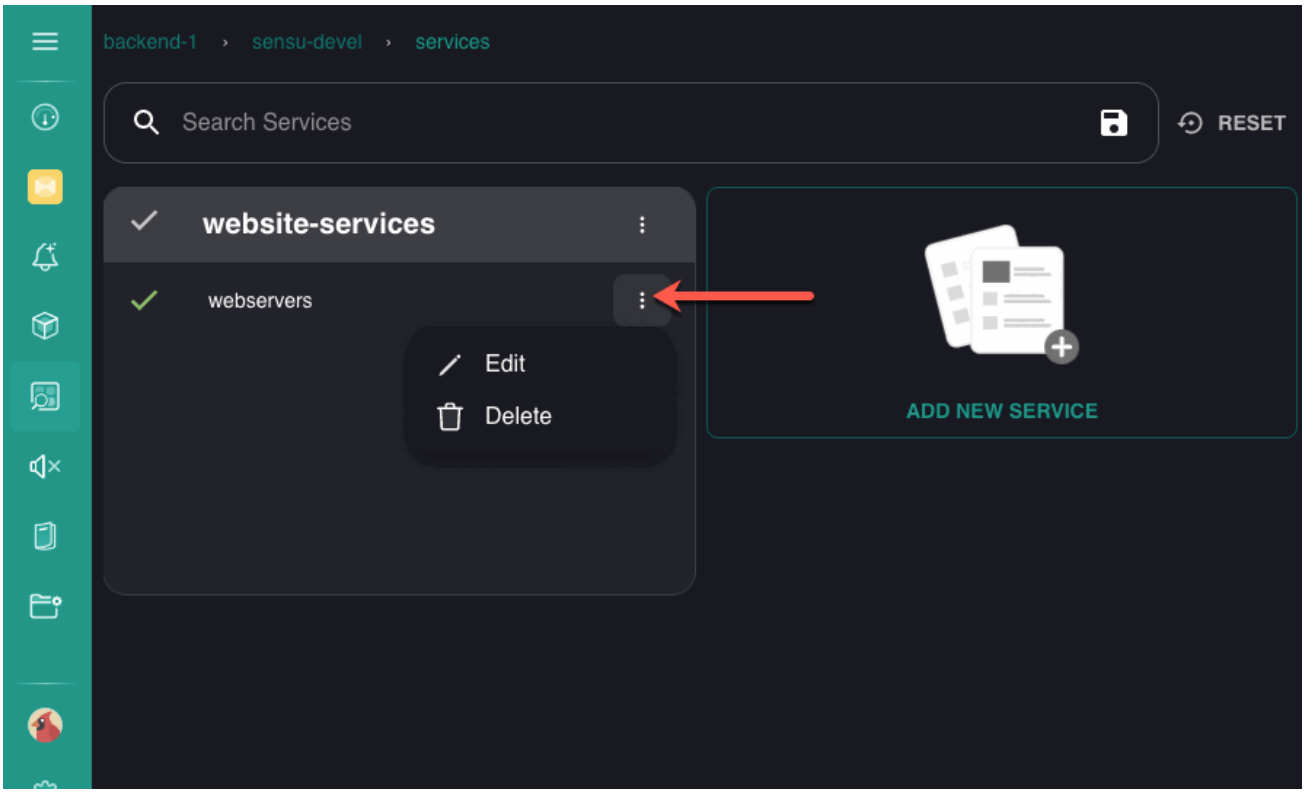
Click the business service name to view its events and other related details and edit, silence, or delete the service:



To edit, add components to, or delete a business service, click ⋮ at the top-right corner of the service's tile.

View and manage service components

After you add a service component to a business service, it will be listed on the business service tile in the web UI Services page until you delete it. To edit or delete a service component, click ⋮ at the right of the component's name:



Click the service component name to view its events and other related details. You can also edit, silence, and delete the component from the detail page.

Searches reference

***COMMERCIAL FEATURE**: Access the web UI in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

With the saved searches feature in the web UI, you can apply search parameters to your entities, events, and resources and save them to etcd in a namespaced resource named `searches`.

The saved searches feature is designed to be used directly in the web UI. However, you can create, retrieve, update, and delete saved searches with enterprise/searches/v1 API endpoints.

Search for events with any status except passing

The following saved search will retrieve all events that have any status except `passing`:

YML

```
---
type: Search
api_version: searches/v1
metadata:
  name: events-not-passing
spec:
  parameters:
  - status:incident
  - status:warning
  - status:critical
  - status:unknown
  resource: core.v2/Event
```

JSON

```
{
  "type": "Search",
  "api_version": "searches/v1",
```

```
  "metadata": {
    "name": "events-not-passing"
  },
  "spec": {
    "parameters": [
      "status:incident",
      "status:warning",
      "status:critical",
      "status:unknown"
    ],
    "resource": "core.v2/Event"
  }
}
```

Search for published checks with a specific subscription and region

The following saved search will retrieve all published checks for the `us-west-1` region with the `linux` subscription:

YML

```
---
type: Search
api_version: searches/v1
metadata:
  name: published-checks-linux-uswest
spec:
  parameters:
  - published:true
  - subscription:linux
  - 'labelSelector: region == "us-west-1"'
  resource: core.v2/CheckConfig
```

JSON

```
{
  "type": "Search",
  "api_version": "searches/v1",
```

```
  "metadata": {
    "name": "published-checks-linux-uswest"
  },
  "spec": {
    "parameters": [
      "published:true",
      "subscription:linux",
      "labelSelector: region == \"us-west-1\""
    ],
    "resource": "core.v2/CheckConfig"
  }
}
```

Search specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For searches in this version of Sensu, the `api_version` should always be `searches/v1`.
required	Required for search entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `api_version: searches/v1` **JSON** `{ "api_version": "searches/v1" }`

metadata

description	Top-level collection of metadata about the search that includes `name` and `namespace`. The `metadata` map is always at the top level of the search definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.
required	Required for search entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML** `metadata:` `  name: us-west-server-incidents` `  namespace: default` **JSON** `{` `  "metadata": {` `  "name": "us-west-server-incidents",` `  "namespace": "default"` `  }` `}`

spec

description	Top-level map that includes the search spec attributes. The spec contents will depend on the search parameters you apply and save.
required	Required for silences in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

example

```
spec:
  parameters:
  - entity:server-testing
  - check:server-health
  - status:incident
  - labelSelector:region == "us-west-1"
  resource: core.v2/Event
```

JSON

```
{
  "spec": {
    "parameters": [
      "entity:server-testing",
      "check:server-health",
      "status:incident",
      "labelSelector:region == \"us-west-1\""
    ],
    "resource": "core.v2/Event"
  }
}
```

type

description	Top-level attribute that specifies the `sensuctl create` resource type. Searches should always be type `Search`.
required	Required for search entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

example

```
type: Search
```

JSON

```
{
  "type": "Search"
}
```

Metadata attributes

name	
description	Search identifier generated from the combination of a subscription name and check name.
required	true
type	String
example	**YML** `name: us-west-server-incidents` **JSON** `{ "name": "us-west-server-incidents" }`

namespace	
description	Sensu RBAC namespace that the search belongs to.
required	false
type	String
default	`default` **YML**

example

```
namespace: default
```

JSON

```
{
  "namespace": "default"
}
```

Spec attributes

parameters	
description	Parameters the search will apply.
required	true
type	Array

YML

example

```
parameters:
- entity:server-testing
- check:server-health
- status:incident
- labelSelector:region == "us-west-1"
```

JSON

```
{
  "parameters": [
    "entity:server-testing",
    "check:server-health",
    "status:incident",
    "labelSelector:region == \"us-west-1\""
  ]
}
```

resource	
description	Fully qualified name of the resource included in the search.
required	true
type	String
example	**YML** `resource: core.v2/Event` **JSON** `{ "resource": "core.v2/Event" }`

Parameters

action	
description	For event filter searches, the type of filter to include in the search: `allow` or `deny`.
required	false
type	String
example	**YML** `parameters:` `- action:allow` **JSON**

```
{
  "parameters": [
    "action:allow"
  ]
}
```

check

description	Name of the check to include in the search.
required	false
type	String

YML

example

```
parameters:
- check:server-health
```

JSON

```
{
  "parameters": [
    "check:server-health"
  ]
}
```

class

description	For entity searches, the entity class to include in the search: `agent` or `proxy`.
required	false
type	String

YML

example

```
parameters:
- class:agent
```

JSON

```
{
  "parameters": [
    "class:agent"
  ]
}
```

entity

description	Name of the entity to include in the search.
required	false
type	String

YML

example

```
parameters:
- entity:server-testing
```

JSON

```
{
  "parameters": [
    "entity:server-testing"
  ]
}
```

event

description	Name of the event to include in the search.
required	false
type	String
example	**YML** `parameters:` `- event:server-testing` **JSON** `{` `  "parameters": [` `  "event:server-testing"` `  ]` `}`

published

description	If `true`, the search will include only published resources. Otherwise, `false`.
required	false
type	Boolean
example	**YML** `parameters:` `- published:true` **JSON** `{` `  "parameters": [` `  "published:true"` `  ]` `}`

silenced

description	If `true`, the search will include only silenced events. Otherwise, `false`.
required	false
type	Boolean
example	**YML** ``` parameters: - silenced:true ``` **JSON** ``` { "parameters": ["silenced:true"] } ```

status

description	Status of the events, entities, or resources to include in the search.
required	false
type	String
example	**YML** ``` parameters: - status:incident ``` **JSON**

```
{
  "parameters": [
    "status:incident"
  ]
}
```

subscription

description	Name of the subscription to include in the search.
required	false
type	String
example	**YML** `parameters:` `- subscription:web` **JSON** `{ "parameters": [ "subscription:web" ] }`

type

description	For handler searches, the type of hander to include in the search: `pipe`, `set`, `tcp`, or `udp`.
required	false
type	String

example

YML

```
parameters:
- type:pipe
```

JSON

```
{
  "parameters": [
    "type:pipe"
  ]
}
```

Web UI configuration reference

***COMMERCIAL FEATURE**: Access web UI configuration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Web UI configuration allows you to define certain display options for the Sensu web UI, such as which web UI theme to use, the number of items to list on each page, and which URLs and linked images to expand. You can define a single custom web UI configuration to federate to all, some, or only one of your clusters.

***NOTE**: Each cluster should have only one web configuration.*

Web UI configuration example

In this web UI configuration example:

- Users will receive a customized sign-in message that is formatted with Markdown
- Details for the local cluster will not be displayed
- Each page will list 50 items (except the checks page, which will list 100 items)
- The web UI will use the classic theme
- The entities page will list only entities with the `proxy` subscription, in ascending order based on `last_seen` value
- The checks page will list checks alphabetically by name
- The web UI will begin to display the license expiration banner 45 days before the organization license expires
- Expanded links and images will be allowed for the listed URLs
- YAML will be the default format for resource definitions in the web UI

YML

```
---
type: GlobalConfig
```

```yaml
api_version: web/v1
metadata:
  name: custom-web-ui
spec:
  signin_message: with your *LDAP or system credentials*
  always_show_local_cluster: false
  catalog:
    disabled: false
    url: "https://catalog.sensu.io"
    release_version: "1.0"
  default_preferences:
    poll_interval: 120000
    page_size: 50
    serialization_format: YAML
    theme: classic
  page_preferences:
  - page: entities
    page_size: 50
    order: LASTSEEN
    selector: proxy in entity.subscriptions
  - page: checks
    page_size: 100
    order: NAME
  license_expiry_reminder: 1080h0m0s
  link_policy:
    allow_list: true
    urls:
    - https://example.com
    - steamapp://34234234
    - "//google.com"
    - "//*.google.com"
    - "//bob.local"
    - https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic
```

JSON

```json
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui"
  },
```

```
  "spec": {
    "signin_message": "with your *LDAP or system credentials*",
    "always_show_local_cluster": false,
    "catalog": {
      "disabled": false,
      "url": "https://catalog.sensu.io",
      "release_version": "1.0"
    },
    "default_preferences": {
      "poll_interval": 120000,
      "page_size": 50,
      "serialization_format": "YAML",
      "theme": "classic"
    },
    "page_preferences": [
      {
        "page": "entities",
        "page_size": 50,
        "order": "LASTSEEN",
        "selector": "proxy in entity.subscriptions"
      },
      {
        "page": "checks",
        "page_size": 100,
        "order": "NAME"
      }
    ],
    "license_expiry_reminder": "1080h0m0s",
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
      ]
    }
  }
}
```

Page preferences order values

Available values for the `order` attribute in page_preferences vary depending on the page.

Page	Order value and description
events	`ENTITY` : List events by the entities that created them, in ascending order by entity name `ENTITY_DESC` : List events by the entities that created them, in descending order by entity name `LASTOK` : List events by their last OK status, starting with the most recent `NEWEST` : List events by their timestamps, starting with the most recent `OLDEST` : List events by their timestamps, starting with the oldest `SEVERITY` : List events by their status, starting with the most severe
entities	`ID` : List entities by their IDs, in ascending order `ID_DESC` : List entities by their IDs, in descending order `LASTSEEN` : List entities by their `last_seen` timestamp, starting with the most recent
silences	`ID` : List silences by their IDs, in ascending order `ID_DESC` : List silences by their IDs, in descending order `BEGIN` : List silences by the time they begin, starting with the silence that begins soonest `BEGIN_DESC` : List silences by the time they begin, ending with the silence that begins first
checks	`NAME` : List checks by name, in alphabetical order `NAME_DESC` : List checks by name, in reverse alphabetical order

event-filters	`NAME` : List event filters by name, in alphabetical order `NAME_DESC` : List event filters by name, in reverse alphabetical order
handlers	`NAME` : List handlers by name, in alphabetical order `NAME_DESC` : List handlers by name, in reverse alphabetical order
mutators	`NAME` : List mutators by name, in alphabetical order `NAME_DESC` : List mutators by name, in reverse alphabetical order

Web UI configuration specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For web UI configuration in this version of Sensu, the api_version should always be `web/v1` .
required	Required for web UI configuration in `wrapped-json` or `yaml` format.
type	String
example	**YML** `api_version: web/v1` **JSON** `{ "api_version": "web/v1" }`

metadata

description	Top-level scope that contains the web UI configuration's `name` and `created_by` information.
required	true
type	Map of key-value pairs
example	**YML** ```yaml metadata: name: custom-web-ui created_by: admin ``` **JSON** ```json { "metadata": { "name": "custom-web-ui", "created_by": "admin" } } ```

spec

description	Top-level map that includes web UI configuration spec attributes.
required	Required for web UI configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs
example	**YML** ```yaml spec: signin_message: with your *LDAP or system credentials* always_show_local_cluster: false catalog: disabled: false url: "https://catalog.sensu.io" ```

```
    release_version: "1.0"
  default_preferences:
    poll_interval: 120000
    page_size: 50
    serialization_format: YAML
    theme: classic
  page_preferences:
    - page: entities
      page_size: 50
      order: LASTSEEN
      selector: proxy in entity.subscriptions
    - page: checks
      page_size: 100
      order: NAME
  license_expiry_reminder: 1080h0m0s
  link_policy:
    allow_list: true
    urls:
    - https://example.com
    - steamapp://34234234
    - "//google.com"
    - "//*.google.com"
    - "//bob.local"
    - https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic
```

JSON

```
{
  "spec": {
    "signin_message": "with your *LDAP or system
credentials*",
    "always_show_local_cluster": false,
    "catalog": {
      "disabled": false,
      "url": "https://catalog.sensu.io",
      "release_version": "1.0"
    },
    "default_preferences": {
      "poll_interval": 120000,
      "page_size": 50,
      "serialization_format": "YAML",
```

```
      "theme": "classic"
    },
    "page_preferences": [
      {
        "page": "entities",
        "page_size": 50,
        "order": "LASTSEEN",
        "selector": "proxy in entity.subscriptions"
      },
      {
        "page": "checks",
        "page_size": 100,
        "order": "NAME"
      }
    ],
    "license_expiry_reminder": "1080h0m0s",
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
      ]
    }
  }
}
```

type	
description	Top-level attribute that specifies the resource type. For web UI configuration, the type should always be `GlobalConfig`.
required	Required for web UI configuration in `wrapped-json` or `yaml` format.

type	String

YML

example	`type: GlobalConfig`

JSON

```
{
  "type": "GlobalConfig"
}
```

Metadata attributes

created_by

description	Username of the Sensu user who created or last updated the web UI configuration. Sensu automatically populates the `created_by` field when the web UI configuration is created or updated. The admin user, cluster admins, and any user with access to the `GlobalConfig` resource can create and update web UI configurations.
required	false
type	String

YML

example	`created_by: admin`

JSON

```
{
  "created_by": "admin"
}
```

name	
description	Name for the web UI configuration that is used internally by Sensu.
required	true
type	String
example	**YML** `name: custom-web-ui` **JSON** `{ "name": "custom-web-ui" }`

Spec attributes

always_show_local_cluster	
description	Use only in federated environments. Set to `true` to display the cluster the user is currently connected to in the namespace switcher. To omit local cluster details, set to `false`.
required	false
type	Boolean
default	`false`
example	**YML** `always_show_local_cluster: false` **JSON** `{`

```
  "always_show_local_cluster": false
}
```

catalog

description	Sensu Catalog configuration preferences. Read Catalog attributes for more information.
required	false
type	Map of key-value pairs
example	**YML** `catalog:` `  disabled: false` `  url: "https://catalog.sensu.io"` `  release_version: "1.0"` **JSON** `{` `  "catalog": {` `  "disabled": false,` `  "url": "https://catalog.sensu.io",` `  "release_version": "1.0"` `  }` `}`

default_preferences

description	Global default preferences page size and theme preferences for all users.
required	false

type	Map of key-value pairs

example

YML

```yaml
default_preferences:
  poll_interval: 120000
  page_size: 50
  theme: classic
```

JSON

```json
{
  "default_preferences": {
    "poll_interval": 120000,
    "page_size": 50,
    "theme": "classic"
  }
}
```

license_expiry_reminder

description

Number of days before license expiration to begin displaying the license expiration banner in the web UI. The value must be a valid duration, such as `1080h`, `14400m`, or `24h59m59s`.

> ***NOTE**: By default, the web UI displays the banner starting 30 days before license expiration.*

required	false
type	String

example

YML

```yaml
license_expiry_reminder: 1080h0m0s
```

JSON

```
{
  "license_expiry_reminder": "1080h0m0s"
}
```

link_policy	
description	For labels or annotations that contain a URL, the policy for which domains are valid and invalid targets for conversion to a link or an image.
required	false
type	Map of key-value pairs
example	**YML** `link_policy:` `  allow_list: true` `  urls:` `  - https://example.com` `  - steamapp://34234234` `  - "//google.com"` `  - "//*.google.com"` `  - "//bob.local"` `  - https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic` **JSON** `{` `  "link_policy": {` `  "allow_list": true,` `  "urls": [` `  "https://example.com",` `  "steamapp://34234234",` `  "//google.com",` `  "//*.google.com",` `  "//bob.local",` `  "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"`

```
    ]
  }
}
```

page_preferences	
description	Page-specific preferences for page size, order, and selector for all users. Any page preferences will override default preferences for the specified page.
required	false
type	Array
example	**YML** `page_preferences:` `  - page: entities` `  page_size: 50` `  order: LASTSEEN` `  selector: proxy in entity.subscriptions` `  - page: checks` `  page_size: 100` `  order: NAME` **JSON** `{` `  "page_preferences": [` `  {` `  "page": "entities",` `  "page_size": 50,` `  "order": "LASTSEEN",` `  "selector": "proxy in entity.subscriptions"` `  },` `  {` `  "page": "checks",` `  "page_size": 100,` `  "order": "NAME"`

```
    }
  ]
}
```

signin_message	
description	Custom message to display on the web UI sign-in modal. Accepts Markdown formatting.
required	false
type	String
default	`with your credentials`
example	**YML** `signin_message: with your *LDAP or system credentials*` **JSON** `{ "signin_message": "with your *LDAP or system credentials*" }`

Catalog attributes

disabled	
description	Set to `true` to disable the Sensu Catalog in the web UI. Otherwise, `false`.
required	false

type	Boolean

YML

example	

```
disabled: false
```

JSON

```
{
  "disabled": false
}
```

release_version

description	Release version to use for generating a private catalog with the Sensu Catalog API.
required	false
type	String

YML

example	

```
release_version: version
```

JSON

```
{
  "release_version": "version"
}
```

url

description	Base URL where Sensu Catalog API output is served for a private catalog. Sensu presents the content published to this URL endpoint in place of the official Sensu Catalog in the web UI.

required	false
type	String
example	**YML** `url: "https://catalog.sensu.io"` **JSON** `{ "url": "https://catalog.sensu.io" }`

Default preferences attributes

page_size	
description	The number of items to list on each page.
required	false
type	Integer
default	`25`
example	**YML** `page_size: 25` **JSON** `{ "page_size": 25 }`

poll_interval

description	The frequency at which web UI pages will poll for new data from the Sensu backend. In milliseconds. Useful for increasing the polling interval duration if web UI sessions are causing heavy load. If you set the poll interval, all web UI views will use the poll interval value instead of their individual polling defaults. ***NOTE**: If an individual user's settings conflict with the web UI configuration settings, Sensu will use the individual user's settings.*
type	Integer
default	`10000` when page is visible. `300000` when page is not visible.
example	**YML** `poll_interval: 120000` **JSON** `{ "poll_interval": 120000 }`

serialization_format

description	Default format for resource definitions in the web UI.
required	false
type	String
default	`YAML`
allowed values	`JSON` , `YAML` **YML**

example

```
serialization_format: YAML
```

JSON

```
{
  "serialization_format": "YAML"
}
```

theme

description

The theme used in the web UI.

> ***NOTE**: If an individual user's settings conflict with the web UI configuration settings, Sensu will use the individual user's settings. For example, if a user's system is set to dark mode and their web UI settings are configured to use their system settings, the web UI will use dark mode for that user, even if you set the theme to `classic` in your web UI configuration.*

required	false
type	String
default	`sensu`
allowed values	`sensu`, `classic`, `uchiwa`, `tritanopia`, `deuteranopia`

example

YML

```
theme: classic
```

JSON

```
{
  "theme": "classic"
}
```

Page preferences attributes

order	
description	The order in which to list items on the specified page. Read Page preferences order values to learn more.
required	false
type	String
example	**YML** `order: LASTSEEN` **JSON** `{ "order": "LASTSEEN" }`

page	
description	The page to which the page preference settings apply.
required	true
type	String
allowed values	`events`, `entities`, `silences`, `checks`, `event-filters`, `handlers`, `mutators`
example	**YML** `page: events`

JSON

```
{
  "page": "events"
}
```

page_size

description	The number of items to list for the specified page.
required	false
type	Integer
example	**YML** `page_size: 100` **JSON** `{ "page_size": 100 }`

selector

description	The search expression to apply to the specified page. ***NOTE**: The selector page preference is not available for the events page.*
required	false

type	String
example	YML ```yaml selector: proxy in entity.subscriptions ``` JSON ```json { "selector": "proxy in entity.subscriptions" } ```

Link policy attributes

allow_list	
description	If the list of URLs acts as an allow list, `true`. If the list of URLs acts as a deny list, `false`. As an allow list, only matching URLs will be expanded. As a deny list, matching URLs will not be expanded, but any other URLs will be expanded.
required	false
type	Boolean
default	`false`
example	YML ```yaml allow_list: true ``` JSON ```json { "allow_list": true } ```

urls	
description	The list of URLs to use as an allow or deny list. ***NOTE**: For images from services that may not have an easily distinguishable file extension, append the anchor `#sensu.io.graphic` to the image URLs.*
required	false
type	Array
example	**YML** `urls:` `- https://example.com` `- steamapp://34234234` `- "//google.com"` `- "//*.google.com"` `- "//bob.local"` `- https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic` **JSON** `{` `  "urls": [` `  "https://example.com",` `  "steamapp://34234234",` `  "//google.com",` `  "//*.google.com",` `  "//bob.local",` `  "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"` `  ]` `}`

Sensu Catalog

***COMMERCIAL FEATURE**: Access the Sensu Catalog in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: The Sensu Catalog is in public preview and is subject to change.*

The Sensu Catalog is a collection of Sensu integrations that provide reference implementations for monitoring and observability and help you integrate Sensu with the platforms and tools you're already using. Catalog integrations are self-service and designed to help you scale up with fewer barriers.

Use the official Sensu Catalog in the web UI

In the official Sensu Catalog in the web UI, users install integrations by following prompts and providing custom information. Sensu then applies any customizations to the integration's resource definitions and deploys the integration configuration to agents in real time.

No external configuration management is required, and users can deploy effective monitoring and observability resources even if they aren't familiar with the Sensu APIs, sensuctl, or the monitoring-as-code workflow.

Read Configure integrations in the Sensu Catalog to learn about the official Sensu Catalog in the web UI.

Create your own catalog of integrations

Instead of using the official Sensu Catalog, you can create a private catalog of custom integrations and make it available to you users within the Sensu web UI.

Read the Catalog integrations reference to learn how to structure your catalog repository, create integration definitions, and use the catalog-api command line interface tool to convert integration files into static API content.

Follow Build a private catalog of Sensu integrations to create your own catalog.

Read the Catalog API documentation to learn more about the requests the catalog-api tool makes to generate the files to display in the Sensu Catalog.

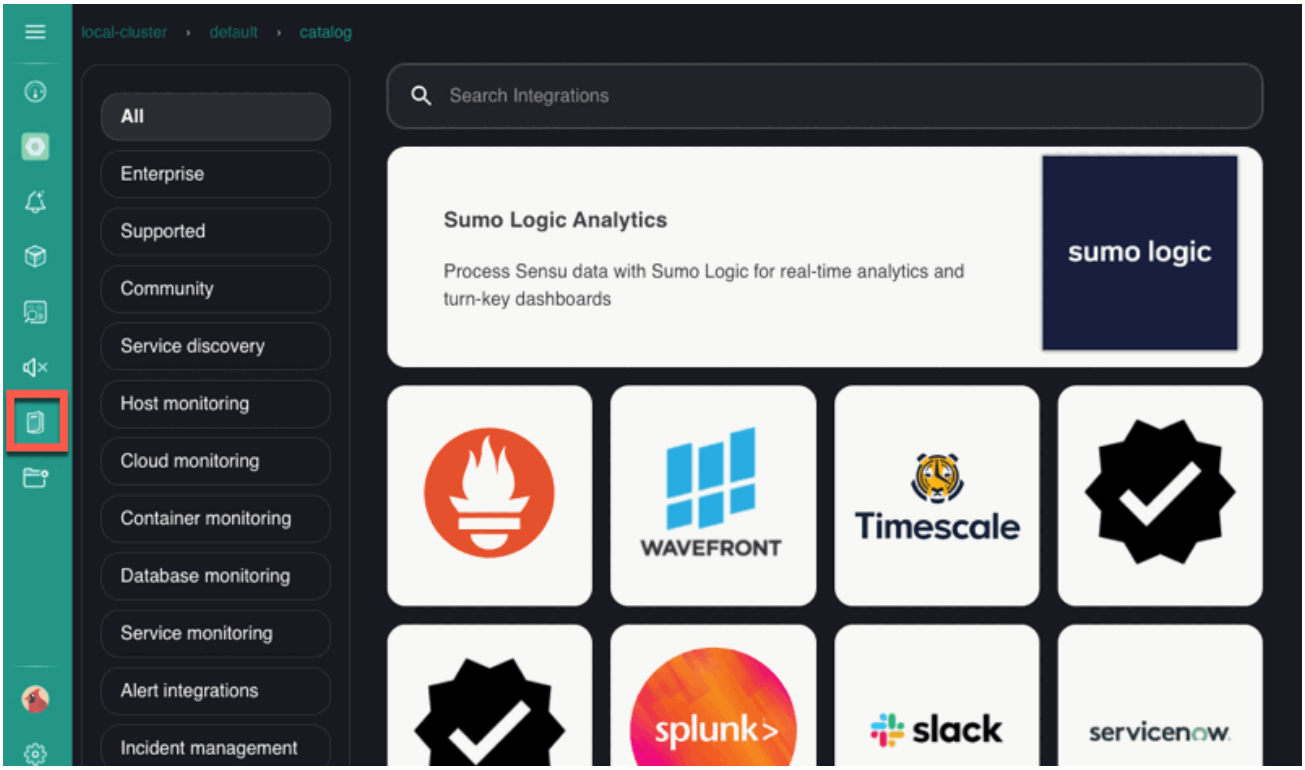
Configure integrations in the Sensu Catalog

***COMMERCIAL FEATURE**: Access the web UI and the Sensu Catalog in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

***NOTE**: The Sensu Catalog is in public preview and is subject to change.*

The Sensu Catalog is an online marketplace for monitoring and observability integrations, from standard system checks and metrics collection to pipelines for sending Sensu data to third-party logging, remediation, and incident management services.

The Sensu Catalog is part of the Sensu web UI, so you can find, configure, and install integrations directly from your browser.



An integration combines a Sensu plugin with a dynamic runtime asset and the Sensu resource definitions that use the plugin.

- The plugin provides the executable script or other program to power a Sensu check, handler, or mutator.
- The dynamic runtime asset is a shareable, reusable package that installs and deploys the plugin.

Integrations provide the plugin and asset along with the recommended or minimum viable configuration and Sensu resources, integrating Sensu with different systems and services for collecting and processing observability data with a few clicks.

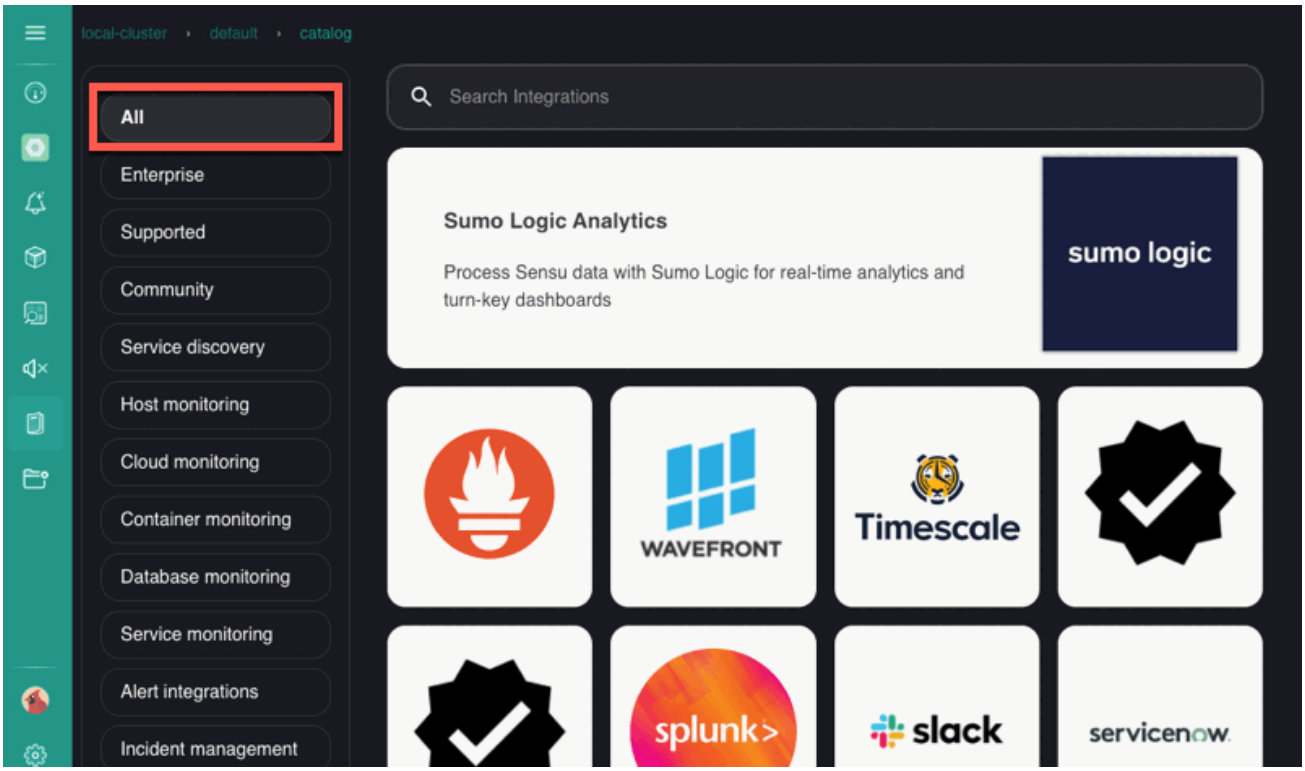
Sensu Catalog integrations allow you to configure powerful real-time monitoring and observability for the systems you rely on. All integrations are self-service and designed to help you scale up with fewer barriers. Sensu curates, tests, and maintains the Catalog integrations, and installation follows a standardized process.

Find integrations

Find integrations in the Sensu Catalog by browsing alphabetized, categorized, and metadata-based lists. You can also search the Sensu Catalog based on integration metadata.

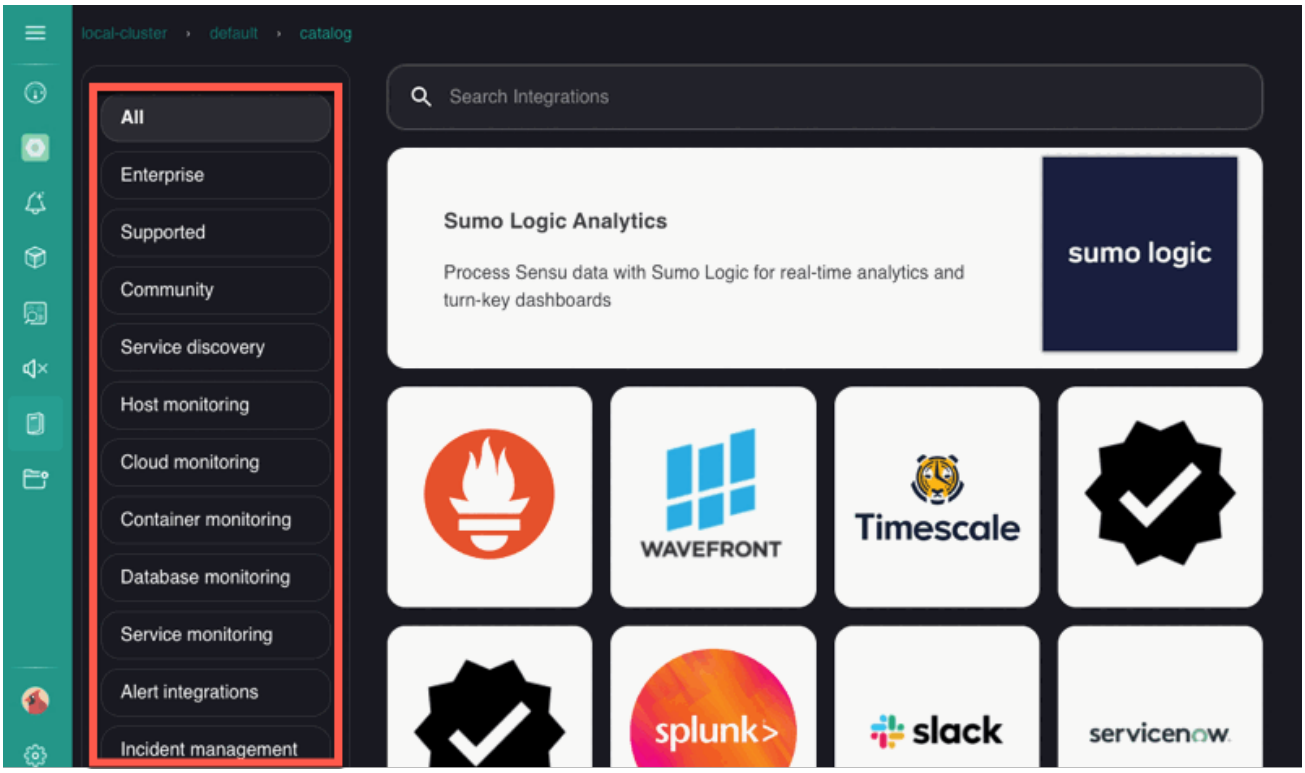
Browse the alphabetized list

When the Catalog page loads in the Sensu web UI, all integrations are alphabetically listed by default. To return to the alphabetized list at any time, click the **All** category in the Catalog page sidebar navigation menu:



Browse the categorized list

The Catalog page sidebar navigation menu lists integrations in categories based on class and function. Click a category to retrieve the associated integrations.



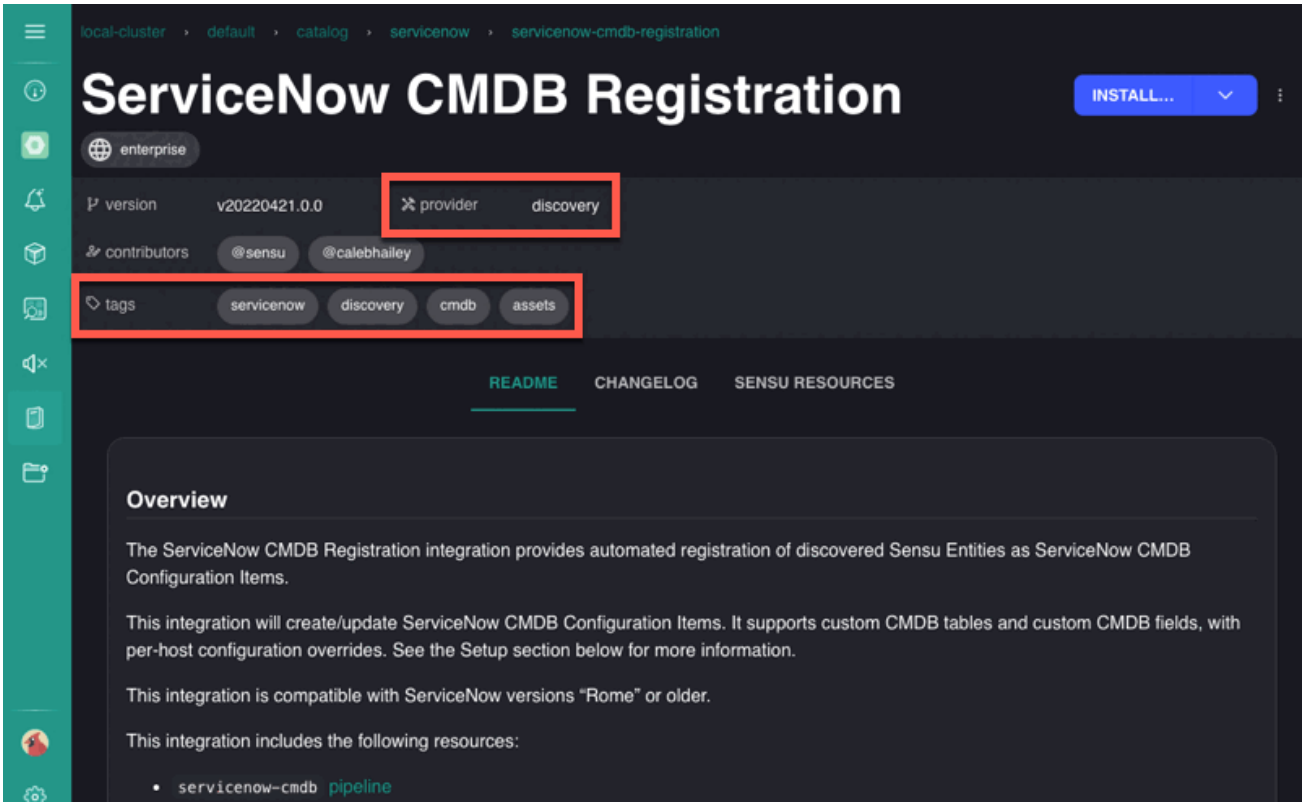
Three categories describe the integration class:

- **Enterprise**: Integrations contributed by one of Sensu's third-party partners.
- **Supported**: Integrations that Sensu developed. Supported integrations may be commercial features that require a valid Sensu license.
- **Community**: Integrations contributed by members of the Sensu community. Community integrations are free and open-source.

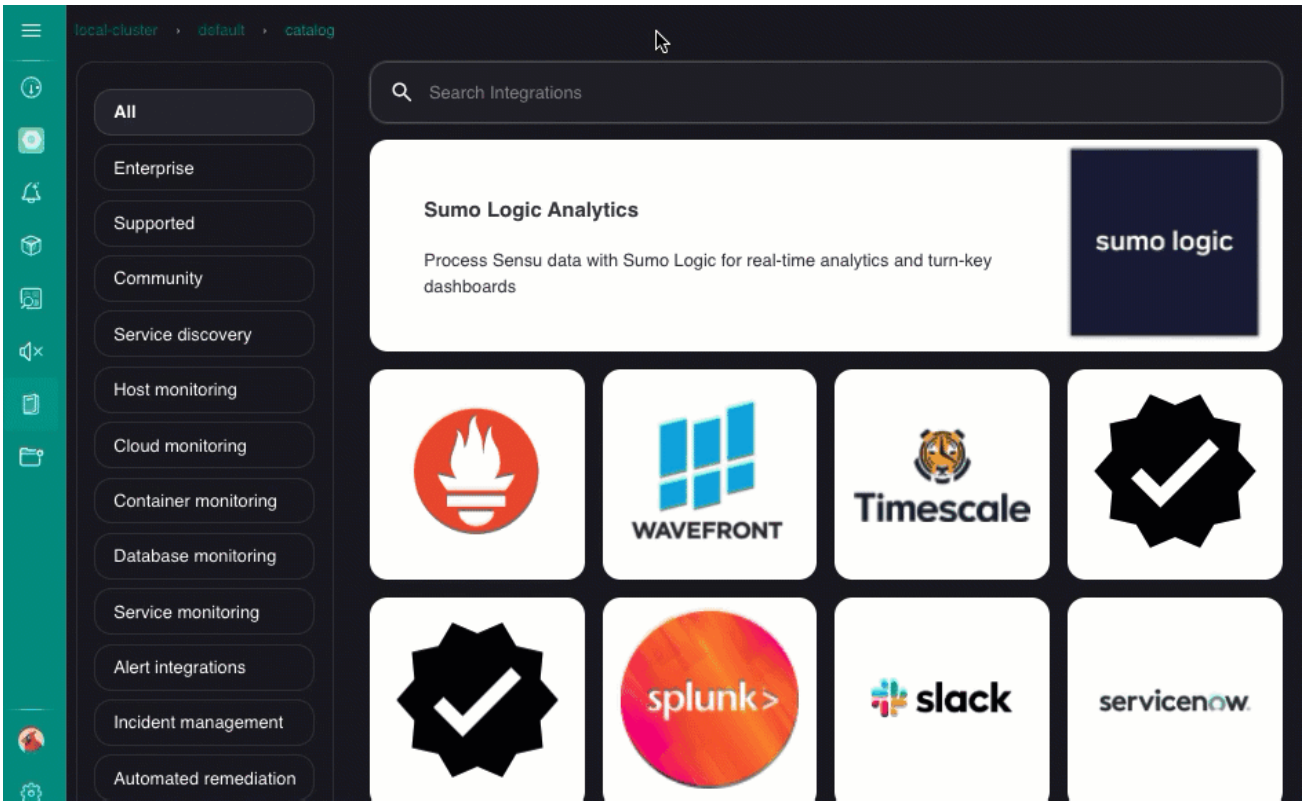
The rest of the categories are based on the integration's function, like cloud monitoring or automated remediation.

Browse a metadata-based list

Each integration has associated metadata listed on the integration detail page:



You can search the Sensu Catalog for integrations with particular `provider` or `tags` metadata from the Catalog main page:



Search for integrations

The Sensu Catalog includes basic search using substring matching, as well as advanced searches based on integration metadata like display name and class.

Catalog search operators

Sensu Catalog search supports two set-based operators:

Operator	Description	Example
`in`	Included in	`ansible in tags`
`matches`	Substring matching	`display_name matches ansible`

Catalog search metadata

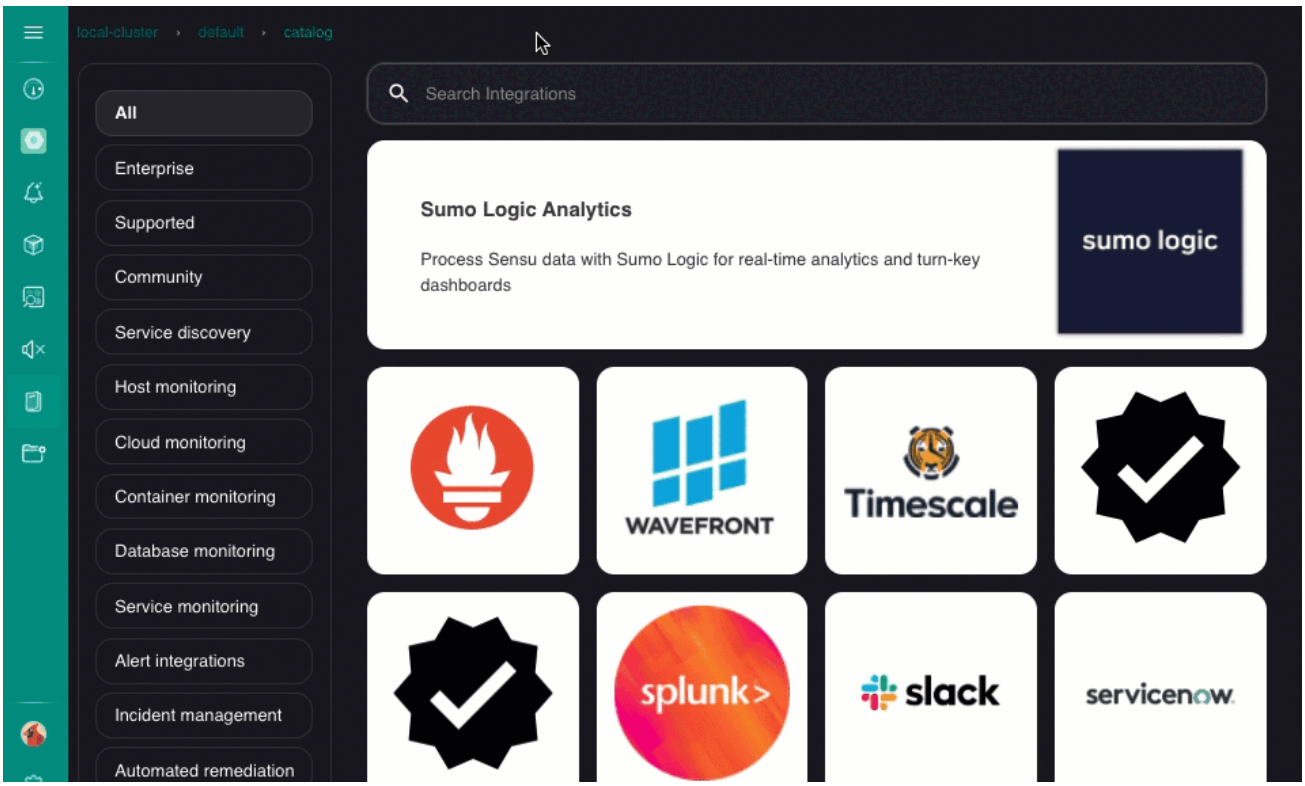
Search the Sensu Catalog integrations based on the following metadata:

Metadata type	Description

`class`	Integration support category. Available values: - `community` : Supported by a Sensu community member - `enterprise` : Supported by Sensu; requires a commercial license - `partner` : Supported by a third-party company or service - `supported` : Supported by Sensu; no license required
`display_name`	Integration name.
`provider`	General function of the integration. Available values: `alerts` , `deregistration` , `discovery` , `events` , `incidents` , `metrics` , `monitoring` , `remediation` .
`tags`	Descriptors added by the integration's creator.

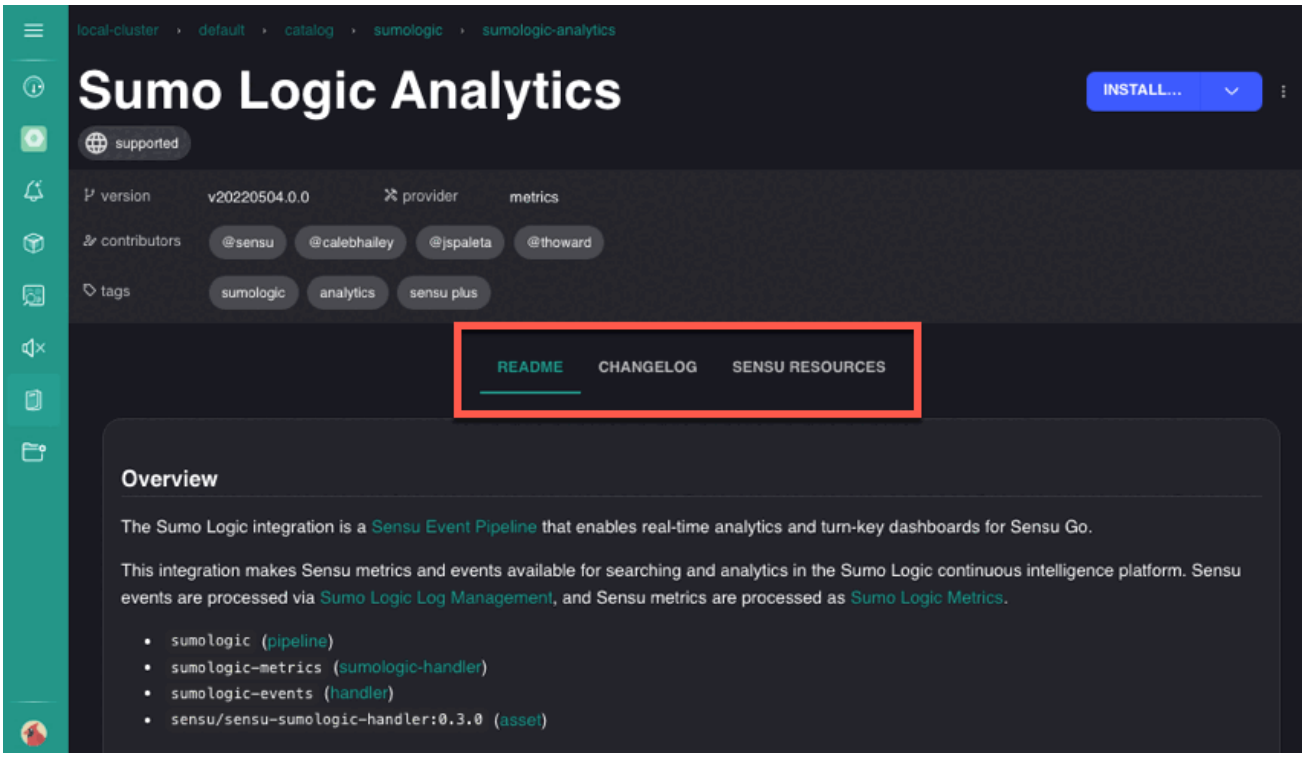
Quick search for integrations

The Sensu Catalog quick search allows you to search without using any particular syntax. Type your search term into the search field on the Catalog page of the web UI and press `Enter` . Sensu will auto-complete a simple search statement for the resources on that page using substring matching:



Get information about an integration

In the Sensu Catalog, integrations are represented by tiles. When you click an integration tile, the integration's detail page opens. The detail page includes tabs for **README**, **CHANGELOG**, and **SENSU RESOURCES**.



The **README** tab contains detailed information about the integration, including an overview, supported dashboards, setup instructions, the plugins the integration requires, the metrics and alerts the integration generates, and links to reference information. The **README** also describes any additional configuration needed to use the integration, like subscriptions to add to agent entities or secrets to create for sensitive information.

The **CHANGELOG** tab lists the notable changes, improvements, and fixes for each version of the integration.

The **SENSU RESOURCES** tab contains usable examples of all of the resource definitions you need to use the integration, including the plugin asset, secrets, checks, handlers, and pipelines. Click the `yaml` or `json` buttons to select the format for each definition.

> ***NOTE**: The SENSU RESOURCES tab lists example resource definitions that you must configure and install. Use the **INSTALL** button to configure and install the integration directly from your browser or copy the example definitions to configure and create with sensuctl or the Sensu API.*

Configure and install an integration

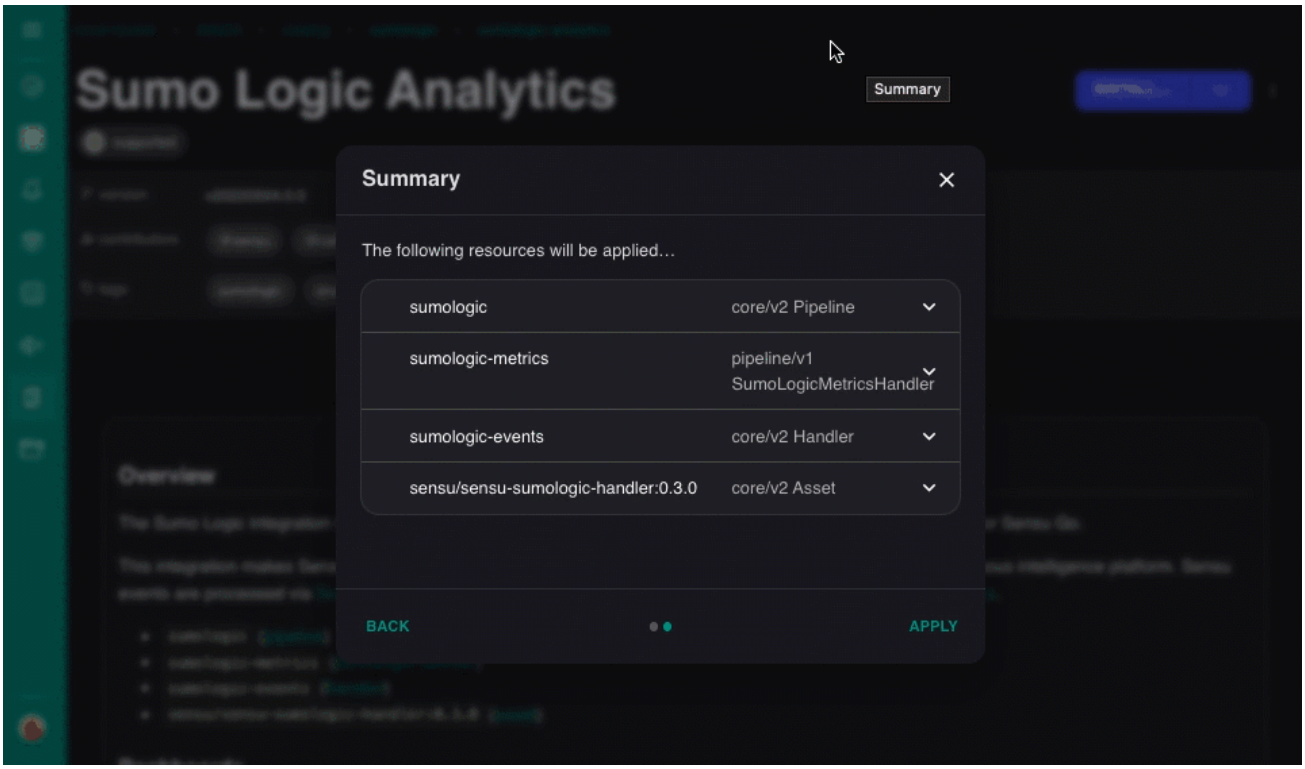
When you find an integration you want to use, click the integration tile to open the detail page. To configure and install an integration:

1. Click **INSTALL** to open the configuration dialog. The configuration dialog is a multi-page form with fields and prompts for collecting additional configuration attributes for the integration.
2. Type values in each attribute field in the dialog to configure the integration for your instance. Use the **NEXT** and **BACK** buttons to navigate through configuration dialog pages as needed.
3. Review the resource definitions on the Summary page.
4. Click **APPLY** to save your configuration and create the integration resources.
5. Click **FINISH** on the confirmation page to close the configuration dialog.

> ***NOTE**: When you click **APPLY** in step 4, Sensu creates all of the resources the integration requires. Check resources are automatically published and will execute immediately.*

The configuration dialog suggests values for each attribute field. These suggestions are collected from your existing resources and refined based on the specific requirements of the integration. For example, if you are setting up a metrics collection integration that requires a pipeline, the dialog will only suggest existing metrics-compatible pipelines for that integration. If you do not have any metrics-compatible pipelines, the dialog will not make suggestions for that attribute.

The Summary page of the configuration dialog lists the definition for each resource that Sensu will create when you click **APPLY**. These resource definitions include the attribute values you provided in the configuration dialog. Click the dropdown arrows to review the resource definitions:



The resulting resource definitions represent Sensu's recommended configuration for the integration.

Use secrets in integrations

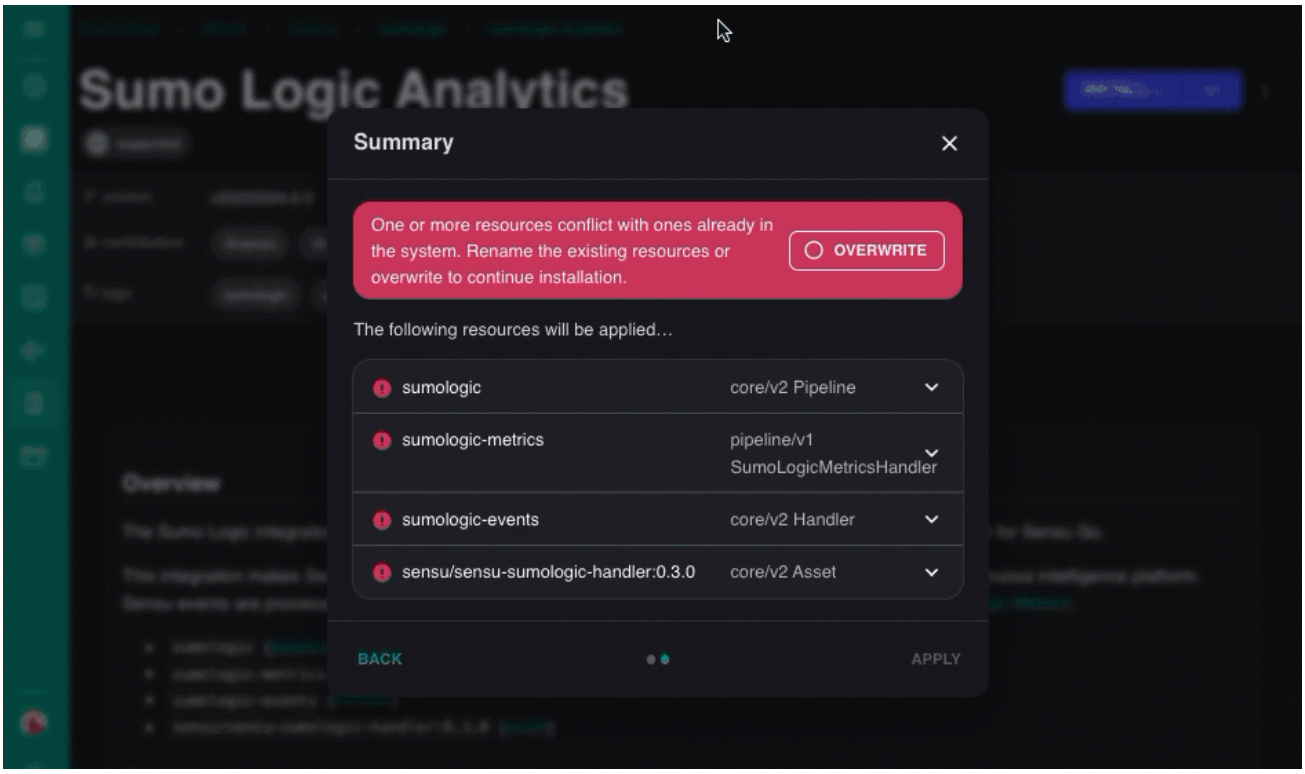
The Sensu Catalog integrations are preconfigured to use Sensu's `Env` secrets provider for sensitive information the integrations might require, like passwords and API tokens.

Duplicate integrations and existing resources

You can reuse the same integration as long as all resource definitions have unique names.

When you install an integration, Sensu checks your existing resources before creating new resources.If Sensu finds an existing resource with the same name, the configuration dialog will prompt you to either change the names of the existing resources or acknowledge that the new resources should overwrite the existing resources.

If you want to keep the existing resources, use the Sensu API to change their names with PUT requests before you continue and create the new resources.Otherwise, click **OVERWRITE** to replace the existing resources with the new resources.



View and manage your integrations

After you install an integration, Sensu creates and publishes the integration resources within your current namespace. The resources are listed on the configuration page for the resource type (checks, filters, handlers, or mutators).

View and manage integration resources just like all of your other Sensu resources: in the web UI, with sensuctl, or with the Sensu API.

Reuse integration resources

The integration definitions listed in the **SENSU RESOURCES** tab are usable, portable definitions for all of the resources you need to use the integration. These definitions are universal monitoring as code templates: they do not include a namespace or the specific values you provide while configuring and installing the integration.

Contribute an integration

The Sensu Catalog is an open marketplace, and you can contribute by sharing Sensu configurations.

For contributing guidelines and more information, visit the Sensu Catalog GitHub repository.

Build a private catalog of Sensu integrations

***COMMERCIAL FEATURE**: Access the Sensu Catalog and integrations in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: The Sensu Catalog is in public preview and is subject to change.*

The Sensu Catalog is a collection of Sensu integrations that provide reference implementations for effective monitoring and observability. The official Sensu Catalog is available in the web UI, but you can also create a private catalog of custom integrations and make it available to users in place of the official Sensu Catalog.

Before you begin, make sure that your integration files are saved in a repository that follows the required organizational framework.

Update URLs in integration asset builds (optional)

***NOTE**: If your catalog assets are stored publicly, you do not need to complete this step. Continue to Install the catalog-api command line tool.*

If the assets for your private catalog are stored behind a firewall or are otherwise not publicly available, update the asset definitions in your `sensu-resources.yaml` files to use the endpoint URL that will serve your catalog.

For example, in the Sensu Catalog repository, asset definitions include `assets.bonsai.sensu.io` in the `builds.url` values:

```
---
type: Asset
api_version: core/v2
metadata:
```

```
  name: sensu/nginx-check
  namespace: default
spec:
  builds:
  - filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'arm'
    - entity.system.arm_version == 6
    headers: null
    sha512:
6471e770fa4232068e1d96b2ad79529483b23dcae109932f095a3d1e59fa22410205c2eb63948e265112
0b217b5bd908856d3cc318af803a45cc531c837a992e
    url:
https://assets.bonsai.sensu.io/02bff14ff7f692daab5cace39dcc6e184751285a/nginx-
check_0.1.0_linux_armv6.tar.gz
  - filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'arm'
    - entity.system.arm_version == 7
    headers: null
    sha512:
714e777c214fd5a7210b67030eb761f5d8c7f8e9ba55f6a0d64872f43f27848eaf51c17bd7b3e3efbdc4
19d4e4754c6143c705b06ddd750009f8068872e5d35d
    url:
https://assets.bonsai.sensu.io/02bff14ff7f692daab5cace39dcc6e184751285a/nginx-
check_0.1.0_linux_armv7.tar.gz
  - filters: ...
```

If assets are not publicly available, replace `assets.bonsai.sensu.io` with your preferred URL in asset `builds.url` values in all `sensu-resources.yaml` files before you continue. You do not need to change the asset `builds.SHA512` values.

Install the catalog-api command line interface tool

The catalog-api command line interface tool is an open-source static API generator: it renders static HTTP API content that the Sensu web UI can consume.

To install the catalog-api tool:

1. Clone the Sensu Catalog API repository and navigate to the local catalog-api repository:

```
git clone https://github.com/sensu/catalog-api && cd catalog-api
```

2. Build the catalog-api tool:

```
go build
```

3. Exit your local copy of the catalog-api repository:

```
cd ..
```

Clone and validate the integration repository

The catalog-api tool consumes content from a repository that includes all the files required to build a catalog of integrations.Follow these steps to clone your repository and validate that all files are organized properly:

1. Clone the repository that stores your Sensu integrations.Replace <REPO_URL> with the URL for your integrations repository uses Sensu's public integration repository:

```
git clone <REPO_URL>
```

2. Navigate to your local copy of the repository that stores the Sensu integrations.Replace <REPO_NAME> with the repository name (for example, for https://github.com/sensu/catalog, the <REPO_NAME> is `catalog`):

```
cd <REPO_NAME>
```

3. Validate the integration repository contents:

```
../catalog-api/catalog-api catalog validate
```

The response lists the integrations found in the local integration repository:

```
11:05AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=path version=99991231.0.0
11:05AM INF Found integration version name=aws-alb-monitoring namespace=aws
source=path version=99991231.0.0
11:05AM INF Found integration version name=aws-ec2-monitoring namespace=aws
source=path version=99991231.0.0
...
11:05AM INF Found integration version name=wavefront-metrics
namespace=wavefront source=path version=99991231.0.0
```

***NOTE**: The catalog-api command line tool also includes server and preview subcommands for viewing your catalog in the web UI during development.*

Generate the private catalog

With a validated repository, you can generate your private catalog locally. The `generate` subcommand generates the static API in a temporary directory, `/tmp/generated-api/`:

```
../catalog-api/catalog-api catalog generate
```

To specify a different temporary directory, use the `--temp-dir` command line flag:

```
../catalog-api/catalog-api catalog generate --temp-dir /tmp/2523661925/release
```

Publish the static API to an endpoint

Once you generate your private catalog in a temporary directory, you can serve the output on any HTTP service and publish it to any endpoint. For example, you can copy the private catalog contents from the temporary directory to a storage service and use a content delivery network (CDN) to serve the content from your storage service to the endpoint URL.

The only requirement is that the endpoint URL must be fetchable for your web UI users. The web UI fetches catalog content from your endpoint; the Sensu backend does not serve any of the catalog content.

Create a UI GlobalConfig definition

Use Sensu's GlobalConfig resource to display the private catalog in the Sensu web UI. Create a GlobalConfig definition that includes the endpoint URL for your private catalog as the `url` value (this example uses `https://catalog.sensu.io:443`):

SHELL

```
cat << EOF | sensuctl create
---
type: GlobalConfig
api_version: web/v1
metadata:
  name: private-catalog
spec:
  always_show_local_cluster: true
  catalog:
    url: "https://catalog.sensu.io:443"
    release_version: version
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "private-catalog"
  },
  "spec": {
```

```
    "always_show_local_cluster": true,
    "catalog": {
      "url": "https://catalog.sensu.io:443",
      "release_version": "version"
    }
  }
}
EOF
```

Confirm the private catalog is available in the web UI

Log into the Sensu web UI at the URL specified in your GlobalConfig resource and navigate to the Catalog page. The Catalog page should include all of the integrations in your repository.

Catalog integrations reference

***COMMERCIAL FEATURE**: Access the Sensu Catalog and integrations in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: The Sensu Catalog is in public preview and is subject to change.*

The Sensu Catalog is a collection of Sensu integrations that provide reference implementations for effective observability. The contents of the official Sensu Catalog are periodically published with the Sensu Catalog API, which is hosted at https://catalog.sensu.io and displayed within the Sensu web UI.

When users install integrations in the Sensu web UI, they receive prompts to enter information. For example, the DNS Monitoring integration includes prompts for the domain name, record type, record class, servers, and port to query. Sensu then applies the user's customizations to the integration's resource definitions and automatically deploys the integration configuration to agents in real time. No external configuration management is required.

Integration definitions resemble other Sensu resources, but Sensu Go does not process them directly. Instead, the catalog-api command line interface tool uses integration definitions along with the other files in the catalog repository, like READMEs and dashboard images, to generate a static Catalog API. The Sensu web UI uses the generated API files to determine which integrations to display in the Sensu Catalog.

The Sensu Catalog provides a way for you and your teams to configure powerful real-time monitoring and observability for the systems you rely on. Integrations are self-service, and the Catalog is designed to help you scale up with fewer barriers.

Integration example

This example shows an integration definition for NGINX monitoring. Integration definitions are saved as the `sensu-integration.yaml` file in a catalog repository:

```
---
api_version: catalog/v1
```

```
type: Integration
metadata:
  namespace: nginx
  name: nginx-monitoring
spec:
  class: supported
  provider: monitoring
  display_name: NGINX Monitoring
  short_description: Monitor NGINX service health and collect metrics
  supported_platforms:
    - darwin
    - linux
    - windows
  tags:
    - http
    - nginx
    - webserver
    - service
  contributors:
    - "@nixwiz"
    - "@calebhailey"
  prompts:
    - type: section
      title: Configure NGINX URL and Monitoring Thresholds
    - type: markdown
      body: |
                Specify the NGINX stub status URL and alerting thresholds for
numbers of active and waiting connections.
    - type: question
      name: default_url
      required: false
      input:
        type: string
        title: NGINX stub status URL
        description: Enter the NGINX stub_status URL
        default: http://127.0.0.1:80/nginx_status
    - type: question
      name: nginx_active_warn
      required: false
      input:
        type: integer
        title: Maximum active connections
```

```
        description: >-
                    Enter the maximum number of active connections to allow before
sending a WARNING event (default is `300`)
        default: 300
    - type: question
      name: nginx_waiting_warn
      required: false
      input:
        type: integer
        title: Maximum waiting connections
        description: >-
                    Enter the maximum number of waiting connections to allow before
sending a WARNING event (default is `30`)
        default: 30
    - type: section
      title: Configure Sensu Subscriptions
    - type: markdown
      body: |
                Specify the subscriptions for Sensu agents that should execute the
`nginx-metrics` check.
    - type: question
      name: subscriptions
      input:
        type: array
        items:
          type: string
          title: Sensu Subscriptions
          ref: core/v2/entity/subscriptions
        default:
          - nginx
    - type: section
      title: Pipeline Configuration
    - type: markdown
      body: |
        Name the [pipelines] you want to use to process NGINX Monitoring integration
data.
        [pipelines]: https://docs.sensu.io/sensu-go/latest/observability-
pipeline/observe-process/pipelines/
    - type: question
      name: alerts_pipeline
      required: false
      input:
```

```
        type: string
        title: Alert pipeline name
        description: >-
                    Which pipeline do you want to use for alerts due to failures
this integration detects?
        ref: core/v2/pipeline/metadata/name
        refFilter: .labels.provider == "alerts"
    - type: question
      name: incidents_pipeline
      required: false
      input:
        type: string
        title: Incident pipeline name
        description: >-
                    Which pipeline do you want to use to process incidents due to
failures this integration detects?
        ref: core/v2/pipeline/metadata/name
        refFilter: .labels.provider == "incidents"
    - type: question
      name: metrics_pipeline
      required: false
      input:
        type: string
        title: Metrics pipeline name
        description: >-
                    Which pipeline do you want to use to process the metrics this
integration collects?
        ref: core/v2/pipeline/metadata/name
        refFilter: .labels.provider == "metrics"
  resource_patches:
    - resource:
        api_version: core/v2
        type: CheckConfig
        name: nginx-metrics
      patches:
        - path: /spec/command
          op: replace
          value: >-
            nginx-check
            --url {{ .annotations.metrics_nginx_url | default "[[ default_url ]]" }}
        - path: /spec/subscriptions
          op: replace
```

```
          value: subscriptions
        - path: /spec/pipelines/-
          op: add
          value:
            api_version: core/v2
            type: Pipeline
            name: "[[metrics_pipeline]]"
        - path: /spec/pipelines/-
          op: add
          value:
            api_version: core/v2
            type: Pipeline
            name: "[[alerts_pipeline]]"
        - path: /spec/pipelines/-
          op: add
          value:
            api_version: core/v2
            type: Pipeline
            name: "[[incidents_pipeline]]"
        - path: /spec/output_metric_thresholds/0/thresholds/0/max
          op: replace
          value: "[[nginx_active_warn]]"
        - path: /spec/output_metric_thresholds/1/thresholds/0/max
          op: replace
          value: "[[nginx_waiting_warn]]"
  post_install:
    - type: section
      title: Success
    - type: markdown
      body: |
        You enabled the NGINX Monitoring integration.
        The `nginx-metrics` check will run for all Sensu agents with these
subscriptions: [[subscriptions]].
```

Catalog repository example

The repository that stores Sensu integrations must organize files in the following structure:

```
integrations/
└── <namespace>/
    └── <integration_name>/
        ├── img/
        │   ├── dashboard-1.gif
        │   └── dashboard-2.png
        ├── CHANGELOG.md
        ├── README.md
        ├── logo.png
        ├── sensu-integration.yaml
        └── sensu-resources.yaml
    └── <integration_name>/
        ├── img/
        │   ├── dashboard-1.gif
        │   └── dashboard-2.png
        ├── CHANGELOG.md
        ├── README.md
        ├── logo.png
        ├── sensu-integration.yaml
        └── sensu-resources.yaml
└── <namespace>/
    └── <integration_name>/
        ├── img/
        │   ├── dashboard-1.gif
        │   └── dashboard-2.png
        ├── CHANGELOG.md
        ├── README.md
        ├── logo.png
        ├── sensu-integration.yaml
        └── sensu-resources.yaml
```

***NOTE**: In the context of catalog integration organization, "namespace" does not refer to the Sensu role-based access control (RBAC) namespace. In catalogs, namespaces are categories for integrations. For example, in the official Sensu Catalog, all integrations for AWS services are organized within the `aws` namespace.*

File	Description
`img`	Images used in the integration README.md, such as screenshots of available dashboards. Image files must be GIF, JPEG, or PNG format.

	External image links are not supported. Optional.
`CHANGELOG.md`	Changelog for the integration. Not displayed in the web UI. Optional.
`README.md`	Help documentation for the integration, including an overview, setup steps, descriptions of the events and metrics the integration produces, and links to supplemental reference information. Sensu supports GitHub-flavored Markdown for integration READMEs. Required.
`logo.png`	Logo image to display in the web UI integration browser. Logo files must be PNG format. Required.
`sensu-integration.yaml`	Metadata for the integration, including title, description, prompts for configuration, patches for updating integration resources, and post-installation instructions. Integration metadata files must be in YAML format and must use the `.yaml` file extension (not `.yml`). Required.
`sensu-resources.yaml`	Sensu resources the integration will install, including checks, handlers, event filters, pipelines, and assets. Do not include RBAC namespaces in the resource definitions in the `sensu-resources.yaml` file. Resources files must be in YAML format and must use the `.yaml` file extension (not `.yml`). Required.

catalog-api command line interface tool

> ***NOTE**: The catalog-api tool is an alpha feature and may include breaking changes.*

Sensu's catalog-api command line interface (CLI) tool generates the static Catalog API to convert integration files into static API content that you can host on any HTTP web service. The Sensu web UI uses the generated API files to determine which integrations to display in the catalog.

Use the catalog-api tool to generate a local Catalog API for testing as you develop new integrations and to build and run a private catalog. Integration files must be stored in a repository that follows the required organizational framework.

The catalog-api tool is written in Go.

catalog-api subcommands

The catalog-api tool provides the following subcommands.

```
catalog-api catalog --help

USAGE
  catalog-api catalog [flags] <subcommand> [flags]

SUBCOMMANDS
  generate  Generate a static catalog API
  validate  Validate a catalog directory and its integrations
  server    Serves static catalog API for development purposes
  preview   Serves static catalog API & preview catalog web application for
development purposes

FLAGS
  -integrations-dir-name integrations  path to the directory containing namespaced
integrations
  -log-level info                      log level of this command ([panic fatal error
warn info debug trace])
  -repo-dir .                          path to the catalog repository
```

Generate subcommand

The generate subcommand generates the contents of a catalog repository locally in a temporary directory, `/tmp/generated-api/`.

Output for the generate subcommand lists the name, catalog namespace, source, and version number for all integration versions:

```
../catalog-api/catalog-api catalog generate

10:40AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=git tag=ansible/ansible-tower-remediation/20220223.0.0
version=20220223.0.0
10:40AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=git tag=ansible/ansible-tower-remediation/20220421.0.0
version=20220421.0.0
10:40AM INF Found integration version name=aws-alb-monitoring namespace=aws
source=git tag=aws/aws-alb-monitoring/20220421.0.0 version=20220421.0.0
```

```
10:40AM INF Found integration version name=aws-ec2-monitoring namespace=aws
source=git tag=aws/aws-ec2-monitoring/20220421.0.0 version=20220421.0.0
...
10:40AM INF Found integration version name=timescaledb-metrics namespace=timescaledb
source=git tag=timescaledb/timescaledb-metrics/20220308.0.0 version=20220308.0.0
10:40AM INF Found integration version name=timescaledb-metrics namespace=timescaledb
source=git tag=timescaledb/timescaledb-metrics/20220421.0.0 version=20220421.0.0
10:40AM INF Found integration version name=wavefront-metrics namespace=wavefront
source=git tag=wavefront/wavefront-metrics/20220421.0.0 version=20220421.0.0
::set-output name=release-
dir::/var/folders/60/cljzzn5n05d91t4x71jx9xzm0000gn/T/3556668713/release
```

The last line of output lists the local path for the generated catalog.

Generate subcommand flags

The catalog-api generate subcommand provides the following configuration flags:

```
catalog-api catalog generate --help

USAGE
  catalog-api catalog generate [flags]

FLAGS
  -integrations-dir-name integrations                          path to the directory
containing namespaced integrations
  -log-level info                                              log level of this
command ([panic fatal error warn info debug trace])
  -repo-dir .                                                  path to the catalog
repository
  -snapshot=false                                              generate a catalog
api for the current catalog branch
  -temp-dir /var/folders/60/cljzzn5n05d91t4x71jx9xzm0000gn/T/  path to a temporary
directory for generated files
  -watch=false                                                 enter watch mode,
which rebuilds on file change
```

Validate subcommand

The validate subcommand confirms that all files in a catalog repository are organized properly.

Output for the validate subcommand lists the name, catalog namespace, source, and version number for integrations found:

```
../catalog-api/catalog-api catalog validate

10:37AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=path version=99991231.0.0
10:37AM INF Found integration version name=aws-alb-monitoring namespace=aws
source=path version=99991231.0.0
10:37AM INF Found integration version name=aws-ec2-monitoring namespace=aws
source=path version=99991231.0.0
...
10:37AM INF Found integration version name=wavefront-metrics namespace=wavefront
source=path version=99991231.0.0
```

Validate subcommand flags

The catalog-api validate subcommand provides the following configuration flags:

```
catalog-api catalog validate --help

USAGE
  catalog-api catalog validate [flags]

FLAGS
  -integrations-dir-name integrations  path to the directory containing namespaced
integrations
  -log-level info                      log level of this command ([panic fatal error
warn info debug trace])
  -repo-dir .                          path to the catalog repository
```

Server subcommand

The server subcommand starts a webserver to serve the JSON files the catalog-api tool generates. To view your catalog in the Sensu web UI while running the server subcommand, you must also configure

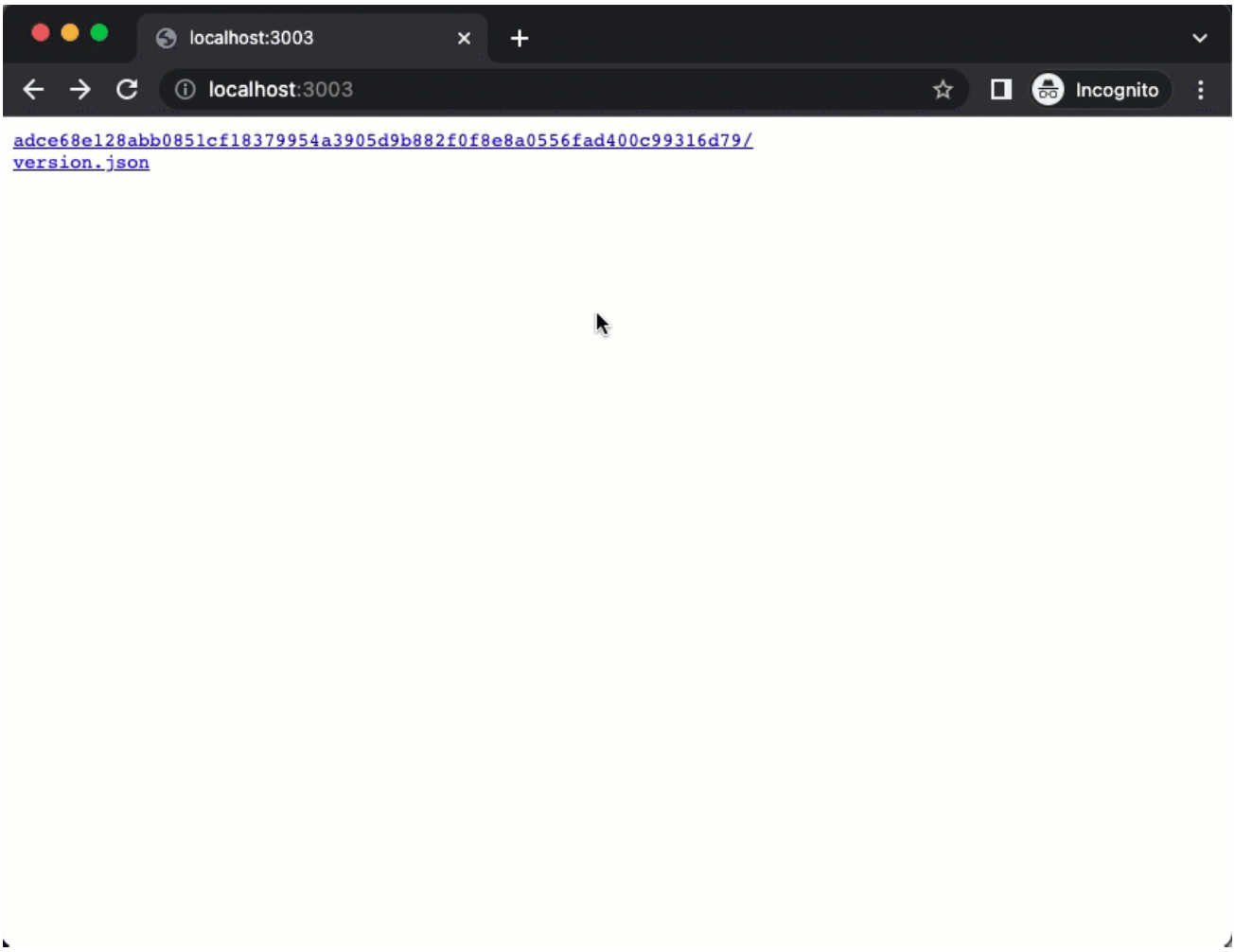
a Sensu backend and create a GlobalConfig resource to point to the webserver.

The last line of the server subcommand response provides the address to use to view the content the catalog-api tool is serving the web UI in your browser. For example:

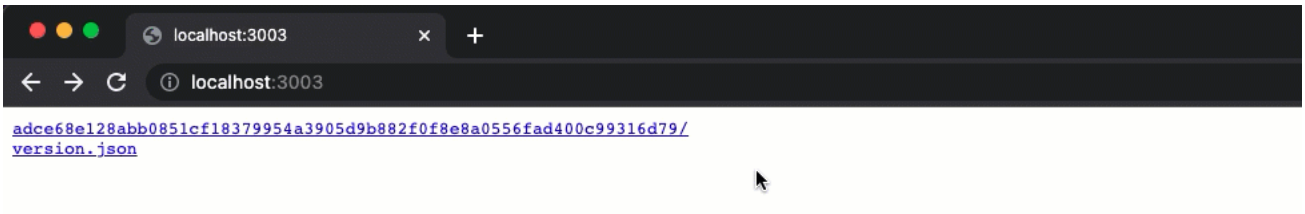
```
10:00AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=git tag=ansible/ansible-tower-remediation/20220223.0.0
version=20220223.0.0
10:00AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=git tag=ansible/ansible-tower-remediation/20220421.0.0
version=20220421.0.0
10:00AM INF Found integration version name=aws-alb-monitoring namespace=aws
source=git tag=aws/aws-alb-monitoring/20220421.0.0 version=20220421.0.0
...
10:00AM INF Found integration version name=wavefront-metrics namespace=wavefront
source=path version=99991231.0.0
10:00AM INF API generated
path=/var/folders/60/cljzzn5n05d91t4x71jx9xzm0000gn/T/2304694052
10:00AM INF API server started address=:3003
```

Visit your webserver address at port 3003 (for example, http://localhost:3003) to view the static Catalog API content that catalog-api is serving.

Click the SHA-256 checksum to view the content for all catalog versions, including the integrations in each catalog version; the JSON definition for each integration version; the catalog repository files for each integration version; and a versions.json file that lists all versions for the integration:



Click version.json to view the contents of the version.json file for the content that catalog-api is serving:



Server subcommand flags

The catalog-api server subcommand provides the following configuration flags:

```
catalog-api catalog server --help

USAGE
  catalog-api catalog server [flags]

FLAGS
```

```
  -integrations-dir-name integrations                              path to the directory
containing namespaced integrations
  -log-level info                                                  log level of this
command ([panic fatal error warn info debug trace])
  -port 8083                                                       port to use for dev
server
  -repo-dir .                                                      path to the catalog
repository
  -temp-dir /var/folders/60/cljzzn5n05d91t4x71jx9xzm0000gn/T/  path to a temporary
directory for generated files
  -watch=false                                                     enter watch mode,
which rebuilds on file change
  -without-snapshot=false                                          generate a catalog
api using tags only
```

Use the Sensu Catalog API server for integration development

When you're developing integrations, it can be helpful to run the Sensu Catalog API server from your local environment so that you can preview integrations as you work. To do this, use the server subcommand in the catalog-api command line tool.

> ***NOTE**: Make sure you have a local Sensu instance running with access to the Sensu web UI.*

1. Clone the Sensu Catalog API repository and navigate to the local catalog-api repository:

   ```
   git clone https://github.com/sensu/catalog-api && cd catalog-api
   ```

2. Build the catalog-api tool:

   ```
   go build
   ```

3. Exit the local catalog-api repository:

   ```
   cd ..
   ```

4. Clone the repository that stores your Sensu integrations. This example uses Sensu's public integration repository:

```
git clone https://github.com/sensu/catalog
```

5. Navigate to your local copy of the repository that stores the Sensu integrations. This example uses https://github.com/sensu/catalog, so the repository is `catalog`:

```
cd ../catalog
```

6. Run the catalog-api server subcommand: This example uses https://github.com/sensu/catalog, so the repository is `catalog`:

```
../catalog-api/catalog-api catalog server --repo-dir . -watch
```

   The `.` in the command tells Sensu to read the catalog contents from your local environment. Use the `-watch` flag to reload the API as you save updates in integration files so that you can see them live in the Sensu web UI.

7. Create a GlobalConfig resource that specifies a local URL for displaying the the private catalog in the Sensu web UI.

   SHELL

```
cat << EOF | sensuctl create
---
type: GlobalConfig
api_version: web/v1
metadata:
  name: private-catalog
spec:
  always_show_local_cluster: true
  catalog:
    url: "https://127.0.0.1:3000"
    release_version: version
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "private-catalog"
  },
  "spec": {
    "always_show_local_cluster": true,
    "catalog": {
      "url": "https://127.0.0.1:3000",
      "release_version": "version"
    }
  }
}
EOF
```

8. Navigate to the Catalog page in the Sensu web UI for your local instance (in this example, https://127.0.0.1:3000). The Catalog page should include all of the integrations in your local repository and update automatically as you save local changes to your integration files.

Preview subcommand

The preview subcommand starts a webserver like the server subcommand but also serves a preview web UI that can communicate with the Sensu backend. If you use the preview subcommand, you do not need to interact with the Sensu backend or create a GlobalConfig resource.

The last line of the preview subcommand response provides the address to use to view the preview catalog in your browser. For example:

```
9:57AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=git tag=ansible/ansible-tower-remediation/20220223.0.0
version=20220223.0.0
9:57AM INF Found integration version name=ansible-tower-remediation
namespace=ansible source=git tag=ansible/ansible-tower-remediation/20220421.0.0
```

```
version=20220421.0.0
9:57AM INF Found integration version name=aws-alb-monitoring namespace=aws
source=git tag=aws/aws-alb-monitoring/20220421.0.0 version=20220421.0.0
...
9:57AM INF Found integration version name=wavefront-metrics namespace=wavefront
source=path version=99991231.0.0
9:57AM INF API generated
path=/var/folders/60/cljzzn5n05d91t4x71jx9xzm0000gn/T/2316699223
9:57AM INF API server started address=:3003
```

Visit your webserver address at port 3003 (for example, http://localhost:3003) to view a preview of the catalog in the Sensu web UI.

Preview subcommand flags

The catalog-api preview subcommand provides the following configuration flags:

```
catalog-api catalog preview --help

USAGE
  catalog-api catalog preview [flags]

FLAGS
  -api-url http://localhost:8080                                    host URL of Sensu
installation; optional
  -integrations-dir-name integrations                               path to the directory
containing namespaced integrations
  -log-level info                                                   log level of this
command ([panic fatal error warn info debug trace])
  -port 3003                                                        port to use for dev
server
  -repo-dir .                                                       path to the catalog
repository
  -temp-dir /var/folders/60/cljzzn5n05d91t4x71jx9xzm0000gn/T/  path to a temporary
directory for generated files
  -without-snapshot=false                                           generate a catalog
api using tags only
  -without-watch=false                                              enter watch mode,
which rebuilds on file change
```

Catalog tags and versions

The catalog-api tool consumes and parses integration-specific git tags to manage and generate versioned integrations. This makes it possible to give users access to earlier versions of integrations and hedge against regressions in individual integrations.

For example, in the official Sensu Catalog repository, two versions of the Ansible Tower Remediation are defined:

```
git tag --list |grep ansible-tower-remediation
ansible/ansible-tower-remediation/20220223.0.0
ansible/ansible-tower-remediation/20220421.0.0
```

Using these tags, the catalog-api tool would generate the following version structure, with both versions of the Ansible Tower Remediation integration:

```
tree /tmp/generated-api/ -L 7
/tmp/generated-api/
├── release
│   ├── 5029648381dff2426ea247147456b4f1227fd6d9050fa42f0660e67a218f8c87
│   │   └── v1
│   │ ├── ansible
│   │ │   ├── ansible-tower-remediation
│   │ │   │   ├── 20220223.0.0
│   │ │   │   │   ├── CHANGELOG.md
│   │ │   │   │   ├── img
│   │ │   │   │   ├── logo.png
│   │ │   │   │   ├── README.md
│   │ │   │   │   └── sensu-resources.json
│   │ │   │   ├── 20220223.0.0.json
│   │ │   │   ├── 20220421.0.0
│   │ │   │   │   ├── CHANGELOG.md
│   │ │   │   │   ├── img
│   │ │   │   │   ├── logo.png
│   │ │   │   │   ├── README.md
│   │ │   │   │   └── sensu-resources.json
│   │ │   │   ├── 20220421.0.0.json
```

```
│   │   │   └── versions.json
│   │   └── ansible-tower-remediation.json
```

Catalog versions

Catalog builds are versioned so that every previous iteration of the catalog is available. You are not limited to providing only the most recent version of the catalog, and you can provide older versions as a fallback.

The catalog-api tool generates builds into a checksum-based output directory structure. The version.json file manages the path to the latest or production catalog API content and instructs the web UI to load catalog contents from the specified checksum directory. When you run the catalog-api generate subcommand to generate the catalog, catalog-api creates the version.json file.

The contents of a version.json file are similar to this example:

```
{
  "release_sha256":
"5029648381dff2426ea247147456b4f1227fd6d9050fa42f0660e67a218f8c87",
  "last_updated": 1655840571
}
```

If you make any changes to your integration files, the catalog-api tool will generate a new checksum directory. To revert to an older build of the catalog, change the `release_sha256` in version.json to point to a different release directory.

Generate version tags

The catalog-api tool uses version tags to create versions of integrations and present them to users within the catalog.

If you update an integration, the first step in publishing the updated integration is to generate a new tag for it:

```
git tag <integration_namespace>/<integration_filename>/<YYYYMMDD>.0.0
```

For example, to generate a new tag for an October 5, 2022 update to the Ansible Tower Remediation integration:

```
git tag ansible/ansible-tower-remediation/20221005.0.0
```

Commit your changes to git after adding the tag. Then, run the catalog-api generate subcommand to generate a catalog that includes the tagged version:

```
../catalog-api/catalog-api catalog generate
```

If you update the integration again on the same day, update the tag to `<YYYYMMDD>.0.1`. To continue the Ansible Tower Remediation example:

```
git tag ansible/ansible-tower-remediation/20221005.0.1
```

Commit your changes to git. The next time you run the catalog-api generate subcommand, it will generate a catalog that includes both tagged versions.

Private catalogs

The catalog-api tool renders static HTTP API content that the Sensu web UI can consume. This means you can create a private enterprise catalog of custom integrations and make it available to users in the Sensu web UI.

You can use the official Sensu Catalog repository, https://github.com/sensu/catalog, as a starting point for building your own private catalog. To do this, clone the repository with the `no-tags` flag to get a copy that does not include Sensu's tags for the existing integrations:

```
git clone --no-tags https://github.com/sensu/catalog
```

The Catalog API defines integrations globally rather than by namespace. When you create a private catalog, all integrations in your repository are available for all users across namespaces in the web UI.

Read Build a private catalog of Sensu integrations for more information.

Integration specification

Top-level attributes

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For integrations in this version of Sensu, the api_version should always be `catalog/v1`.
required	true
type	String
example	`api_version: catalog/v1`

metadata	
description	Top-level scope that contains the integration's `name` and `namespace` information.
required	true
type	Map of key-value pairs
example	`metadata:` `  namespace: nginx` `  name: nginx-monitoring`

spec

description	Top-level map that includes integration spec attributes.
required	true
type	Map of key-value pairs
example	

```
spec:
  class: supported
  provider: monitoring
  display_name: NGINX Monitoring
  short_description: Monitor NGINX service health and
collect metrics
  supported_platforms:
    - darwin
    - linux
    - windows
  tags:
    - http
    - nginx
    - webserver
    - service
  contributors:
    - "@nixwiz"
    - "@calebhailey"
  prompts:
    - type: section
      title: Configure NGINX URL and Monitoring Thresholds
    - type: markdown
      body: |
                Specify the NGINX stub status URL and
alerting thresholds for numbers of active and waiting
connections.
    - type: question
      name: default_url
      required: false
      input:
        type: string
        title: NGINX stub status URL
        description: Enter the NGINX stub_status URL
        default: http://127.0.0.1:80/nginx_status
    - type: question
      name: nginx_active_warn
```

```
      required: false
      input:
        type: integer
        title: Maximum active connections
        description: >-
                    Enter the maximum number of active
connections to allow before sending a WARNING event
(default is `300`)
        default: 300
    - type: question
      name: nginx_waiting_warn
      required: false
      input:
        type: integer
        title: Maximum waiting connections
        description: >-
                    Enter the maximum number of waiting
connections to allow before sending a WARNING event
(default is `30`)
        default: 30
    - type: section
      title: Configure Sensu Subscriptions
    - type: markdown
      body: |
                Specify the subscriptions for Sensu agents
that should execute the `nginx-metrics` check.
    - type: question
      name: subscriptions
      input:
        type: array
        items:
          type: string
          title: Sensu Subscriptions
          ref: core/v2/entity/subscriptions
        default:
          - nginx
    - type: section
      title: Pipeline Configuration
    - type: markdown
      body: |
        Name the [pipelines] you want to use to process
NGINX Monitoring integration data.
```

```
        [pipelines]: https://docs.sensu.io/sensu-
go/latest/observability-pipeline/observe-process/pipelines/
    - type: question
      name: alerts_pipeline
      required: false
      input:
        type: string
        title: Alert pipeline name
        description: >-
                    Which pipeline do you want to use for
alerts due to failures this integration detects?
        ref: core/v2/pipeline/metadata/name
        refFilter: .labels.provider == "alerts"
    - type: question
      name: incidents_pipeline
      required: false
      input:
        type: string
        title: Incident pipeline name
        description: >-
                    Which pipeline do you want to use to
process incidents due to failures this integration detects?
        ref: core/v2/pipeline/metadata/name
        refFilter: .labels.provider == "incidents"
    - type: question
      name: metrics_pipeline
      required: false
      input:
        type: string
        title: Metrics pipeline name
        description: >-
                    Which pipeline do you want to use to
process the metrics this integration collects?
        ref: core/v2/pipeline/metadata/name
        refFilter: .labels.provider == "metrics"
  resource_patches:
    - resource:
        api_version: core/v2
        type: CheckConfig
        name: nginx-metrics
      patches:
        - path: /spec/command
```

```
          op: replace
          value: >-
            nginx-check
            --url {{ .annotations.metrics_nginx_url |
default "[[ default_url ]]" }}
        - path: /spec/subscriptions
          op: replace
          value: subscriptions
        - path: /spec/pipelines/-
          op: add
          value:
            api_version: core/v2
            type: Pipeline
            name: "[[metrics_pipeline]]"
        - path: /spec/pipelines/-
          op: add
          value:
            api_version: core/v2
            type: Pipeline
            name: "[[alerts_pipeline]]"
        - path: /spec/pipelines/-
          op: add
          value:
            api_version: core/v2
            type: Pipeline
            name: "[[incidents_pipeline]]"
        - path:
/spec/output_metric_thresholds/0/thresholds/0/max
          op: replace
          value: "[[nginx_active_warn]]"
        - path:
/spec/output_metric_thresholds/1/thresholds/0/max
          op: replace
          value: "[[nginx_waiting_warn]]"
  post_install:
    - type: section
      title: Success
    - type: markdown
      body: |
        You enabled the NGINX Monitoring integration.
        The `nginx-metrics` check will run for all Sensu
agents with these subscriptions: [[subscriptions]].
```

type	
description	Top-level attribute that specifies the resource type. For integrations, the type should always be `Integration`.
required	true
type	String
example	`type: Integration`

Metadata attributes

name	
description	Name for the integration that is used internally by Sensu.
required	true
type	String
example	`name: nginx-monitoring`

namespace	
description	Sensu RBAC namespace that the integration belongs to.
required	false
type	String
example	

```
namespace: nginx
```

Spec attributes

class	
description	Class to use for categorizing the integration in the web UI.
required	true
type	String
allowed values	¬ `community` for community-supported integrations ¬ `supported` for Sensu-supported integrations ¬ `enterprise` for Sensu-supported integrations that require a commercial license ¬ `partner` for integrations supported by Sensu's third-party partners
example	`class: community`

contributors	
description	List of GitHub @usernames to display on integration detail pages in the web UI.
required	true
type	Array
example	`contributors:` `  - "@nixwiz"`

```
  - "@calebhailey"
```

display_name

description	Name to display for the integration in the web UI.
required	true
type	String
example	`display_name: NGINX Monitoring`

post_install

description	Content to display for the final step in integration configuration. The post_install dialog is helpful for confirming successful installation and providing instructions for any further configuration an integration may require. If you do not include a post_install array in your integration definition, Sensu will display a default “Success” window. Read Post install attributes for more information.
required	false
type	Array
example	`post_install:` `  - type: section` `  title: Success` `  - type: markdown` `  body: |` `  You enabled the NGINX Monitoring integration.` `  The \`nginx-metrics\` check will run for all Sensu agents with these subscriptions: [[subscriptions]].`

prompts	
description	Attributes for soliciting user-provided variable values to use in `resource_patches`. Read Prompts attributes for more information.
required	true
type	Map of key-value pairs
example	

```
prompts:
  - type: section
    title: Configure NGINX URL and Monitoring Thresholds
  - type: markdown
    body: |
            Specify the NGINX stub status URL and alerting
thresholds for numbers of active and waiting connections.
  - type: question
    name: default_url
    required: false
    input:
      type: string
      title: NGINX stub status URL
      description: Enter the NGINX stub_status URL
      default: http://127.0.0.1:80/nginx_status
  - type: question
    name: nginx_active_warn
    required: false
    input:
      type: integer
      title: Maximum active connections
      description: >-
                Enter the maximum number of active
connections to allow before sending a WARNING event
(default is `300`)
      default: 300
  - type: question
    name: nginx_waiting_warn
    required: false
    input:
      type: integer
```

```
      title: Maximum waiting connections
      description: >-
                Enter the maximum number of waiting
connections to allow before sending a WARNING event
(default is `30`)
      default: 30
  - type: section
    title: Configure Sensu Subscriptions
  - type: markdown
    body: |
            Specify the subscriptions for Sensu agents that
should execute the `nginx-metrics` check.
  - type: question
    name: subscriptions
    input:
      type: array
      items:
        type: string
        title: Sensu Subscriptions
        ref: core/v2/entity/subscriptions
      default:
        - nginx
  - type: section
    title: Pipeline Configuration
  - type: markdown
    body: |
      Name the [pipelines] you want to use to process NGINX
Monitoring integration data.
      [pipelines]: https://docs.sensu.io/sensu-
go/latest/observability-pipeline/observe-process/pipelines/
  - type: question
    name: alerts_pipeline
    required: false
    input:
      type: string
      title: Alert pipeline name
      description: >-
                Which pipeline do you want to use for
alerts due to failures this integration detects?
      ref: core/v2/pipeline/metadata/name
      refFilter: .labels.provider == "alerts"
  - type: question
```

```
    name: incidents_pipeline
    required: false
    input:
      type: string
      title: Incident pipeline name
      description: >-
                Which pipeline do you want to use to
process incidents due to failures this integration detects?
      ref: core/v2/pipeline/metadata/name
      refFilter: .labels.provider == "incidents"
  - type: question
    name: metrics_pipeline
    required: false
    input:
      type: string
      title: Metrics pipeline name
      description: >-
                Which pipeline do you want to use to
process the metrics this integration collects?
      ref: core/v2/pipeline/metadata/name
      refFilter: .labels.provider == "metrics"
```

provider

description	Integration function to use for categorizing the integration in the web UI.
required	true
type	String
allowed values	`alerts`, `deregistration`, `discovery`, `events`, `incidents`, `metrics`, `monitoring`, `remediation`
example	`provider: monitoring`

resource_patches

description	Attributes that define how to apply changes to the integration resources in the `sensu-resources.yaml` file based on user responses to prompts. Read Resource patches attributes for more information.
required	true
type	Map of key-value pairs
example	

```
resource_patches:
  - resource:
      api_version: core/v2
      type: CheckConfig
      name: nginx-metrics
    patches:
      - path: /spec/command
        op: replace
        value: >-
          nginx-check
          --url {{ .annotations.metrics_nginx_url | default
"[[ default_url ]]" }}
      - path: /spec/subscriptions
        op: replace
        value: subscriptions
      - path: /spec/pipelines/-
        op: add
        value:
          api_version: core/v2
          type: Pipeline
          name: "[[metrics_pipeline]]"
      - path: /spec/pipelines/-
        op: add
        value:
          api_version: core/v2
          type: Pipeline
          name: "[[alerts_pipeline]]"
      - path: /spec/pipelines/-
        op: add
        value:
          api_version: core/v2
          type: Pipeline
          name: "[[incidents_pipeline]]"
```

```
      - path:
/spec/output_metric_thresholds/0/thresholds/0/max
        op: replace
        value: "[[nginx_active_warn]]"
      - path:
/spec/output_metric_thresholds/1/thresholds/0/max
        op: replace
        value: "[[nginx_waiting_warn]]"
```

short_description

description	Brief description of the integration to display in the web UI.
required	true
type	String
example	`short_description: Monitor NGINX service health and collect metrics`

supported_platforms

description	Supported platforms for the integration. Used for checks only.
required	true
type	Array

example

```
supported_platforms:
  - darwin
  - linux
  - windows
```

tags

description	Keywords for the integration. Used for integration searches in the web UI.
required	true
type	Array
example	`tags:` `  - http` `  - nginx` `  - webserver` `  - service`

Post install attributes

body	
description	Markdown content to display in the integration post install dialog. If you specify `type: markdown`, you must provide a `body` attribute.
required	false
type	String
example	`body: |` `  You enabled the NGINX Monitoring integration.` `  The `nginx-metrics` check will run for all Sensu agents with these subscriptions: [[subscriptions]].`

title	
description	Section title to display in the integration post install dialog. If you specify `type: section`, you must provide a `title` attribute.
required	false

type	String
example	`title: Success`

type

description	Type of post install content to display. To configure a window of post install content, include a `type: section` attribute and a `type: markdown` attribute. For `type: section`, provide a title. For `type: markdown`, provide a body. Each `type: section` attribute you add corresponds to one window of post install content; if you need more than one window of post install content, add another `type: section` attribute.
required	false
type	String
example	`type: section`

Prompts attributes

body

description	Markdown content to display in a prompt. If you specify `type: markdown`, you must include a `body` attribute. Body attributes are useful for providing instructions at the top of each prompt window.
required	false
type	String
example	

```
body: |
    Specify the NGINX stub status URL and alerting
thresholds for numbers of active and waiting connections.
```

input

description	Configuration attributes for `type: question` prompts. Read Input attributes for more information.
required	false
type	Map of key-value pairs
example	`input:` `  type: string` `  title: NGINX stub status URL` `  description: Enter the NGINX stub_status URL` `  default: http://127.0.0.1:80/nginx_status`

name

description	Variable name for use as a reference in resource patches to substitute user input for a specific attribute's value in an integration resource. You can also interpolate integration variable names into string templates with double square brackets (e.g. `Hello, [[ team ]]!`). Used with `type: question` prompts.
required	false
type	String
example	`name: default_url`

required

description	If the associated prompt requires user input, `true`. Otherwise, `false`. Used with `type: question` prompts.
required	false
type	Boolean
example	`attribute:`

title

description	Section title to display in the integration prompts dialog. If you specify `type: section`, you must provide a `title` attribute.
required	false
type	String
example	`title: Configure NGINX URL and Monitoring Thresholds`

type

description	Type of prompt to display. To configure a window of prompts, include a `type: section` attribute followed by a title. Within each window of prompts, use `type: question` attributes to collect user responses and `type: markdown` attributes to provide user instructions. Each `type: section` attribute you add corresponds to one window of prompts; if you need more than one window of prompts, add another `type: section` attribute.
required	false
type	String

example	`type: section`

Resource patches attributes

patches	
description	Updates to apply to the selected resource, in JSON Patch format. Variable substitution and templating are supported with `varname` references in double square brackets (for example, `Hello, [[varname]]`). If an individual operation fails, Sensu considers it optional and skips it. All patches must specify a `path`, `op` (operation), and `value`. Read Patches attributes for more information.
required	false
type	Map of key-value pairs
example	`patches:` `  - path: /spec/command` `  op: replace` `  value: >-` `  nginx-check` `  --url {{ .annotations.metrics_nginx_url | default "[[ default_url ]]" }}` `  - path: /spec/subscriptions` `  op: replace` `  value: subscriptions`

resource	
description	Identification information for the Sensu API resource to patch. The

	resource must be included in the integration's `sensu-resources.yaml` file. Read Resource attributes for more information.
required	false
type	Map of key-value pairs
example	`- resource:` `  api_version: core/v2` `  type: CheckConfig` `  name: nginx-metrics`

Input attributes

default

description	Default value to use for the associated attribute if the user does not specify a value.
required	false
type	String
example	`default: http://127.0.0.1:80/nginx_status`

description

description	Description to display below the user input field.
required	false
type	String
example	`description: Enter the NGINX stub_status URL`

format

description	Format for the input value. Some display formats provide helpers that simplify user input.
required	false
type	String
allowed values	`cron`, `duration` `ecmascript-5.1`, `email`, `envvar`, `hostname`, `io.sensu.selector`, `ipv4`, `ipv6`, `tel`, `url`, `sh`, `sha-256`, `sha-512`
example	`format: email`

ref

description	Reference to a Sensu API resource in the format `<api_group>/<version>/<api_resource>/<api_field_path>` (for example, `core/v2/pipelines/metadata/name` refers to the names of core/v2 pipelines resources). The referenced resources are presented to the user in a drop-down selector. Sensu captures the resource the user selects as the input value.
required	false
type	String
example	`ref: core/v2/entity/subscriptions`

refFilter

description	Filters to apply to Sensu API resource references in Sensu Query

	Expression (SQE) format. Sensu uses `refFilter` values to filter `ref` results.
required	false
type	String
example	`refFilter: .labels.provider == "alerts"`

title

description	Label to display above the user input field.
required	true
type	String
example	`title: NGINX stub status URL`

type

description	Type of input requested.
required	true
type	String
allowed values	`boolean`, `integer`, `string`
example	`type: string`

Patches attributes

op

description	Patch operation to perform.
required	false
type	String
allowed values	`add`, `replace`
example	`op: replace`

path

description	Path for the attribute to patch within the specified Sensu resource. In JSON Pointer format, which supports array indexes such as `/spec/subscriptions/0`. Use `-` to insert values at the end of an array (for example, `/spec/subscriptions/-`).
required	false
type	String
example	`path: /spec/subscriptions`

value

description	Built-in or user-entered value to apply in the patch. The built-in value is `unique_id`, which randomly generates an 8-digit hexadecimal string value (e.g. 168c41a1). User-entered variables are represented by variable name references in double square brackets.
required	false
type	
example	

```
value: [[ subscriptions ]]
```

Resource attributes

api_version	
description	Sensu API group and version for the resource to patch.
required	true
type	String
example	`api_version: core/v2`

name	
description	Name of the resource to patch.
required	true
type	String
example	`name: nginx-metrics`

type	
description	Type of the resource to patch.
required	true
type	String
example	`type: CheckConfig`

Resource limits

There is no limit on the number of resources you can bundle into a single integration. Each integration can include as many checks, event filters, handlers, and pipelines as you need to achieve your observability goals. For example, you can develop a single host monitoring integration that installs all of the checks you want to run on every server.

Check guidelines

For integrations that create checks, list the resource definitions in your `sensu-resources.yaml` file in the following order:

1. CheckConfig
2. HookConfig
3. Secret
4. Asset

The `sensu-integration.yaml` file for check resources should include at least one subscription, whether it is provided as a default or requested with a prompt. Subscriptions should be named according to the check's function. For example, a PostgreSQL monitoring check could include a subscription named `postgresql`.

Use the YAML `>-` multiline block scalar syntax to wrap the check `command` value and make it easier to read. For example:

```
spec:
  command: >-
    check-disk-usage.rb
    -w {{ .annotations.disk_usage_warning | default 85 }}
    -c {{ .annotations.disk_usage_critical | default 95 }}
```

Use tunables like tokens in your check commands as needed, sourced from entity annotations (not labels) and with explicitly configured default values.

Check resources should use interval scheduling with a minimum interval of 30 seconds. Set check

`timeout` to a non-zero value that is no greater than 50% of the interval.

Prompts for check pipelines should use one of the following generic categories:

- Alerts
- Incident management
- Metrics
- Events
- Deregistration
- Remediation

Pipeline guidelines

For integrations that create pipelines, list the resource definitions in your `sensu-resources.yaml` file in the following order:

1. Pipeline
2. Handler, SumoLogicMetricsHandler, and TCPStreamHandler
3. Filter
4. Mutator
5. Secret
6. Asset

For alert and incident management pipelines, we recommend using the built-in is_incident and not_silenced event filters instead of custom event filters that are configured for specific use cases.

Asset guidelines

Asset resources and their corresponding `runtime_assets` references in other Sensu resources must include an asset version reference in their resource name. For example, `sensu/system-check:0.5.0`.

Asset resources should include an organization or author as the namespace in the resource name. For example, the official Sensu PagerDuty plugin hosted in the `sensu` organization on GitHub (sensu/sensu-pagerduty-handler) and published to under the `sensu` organization on Bonsai (sensu/sensu-pagerduty-handler) should be named `sensu/sensu-pagerduty-handler`.

For integrations contributed to the official Sensu Catalog, asset resources in the `sensu-`

`resources.yaml` file must refer to assets hosted on Bonsai.Read Build a private catalog of Sensu integrations for information about using assets that are stored behind a firewall or are otherwise not publicly available.

Catalog API

***COMMERCIAL FEATURE**: Access the Catalog API in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: The Sensu Catalog is in public preview and is subject to change.*

The Catalog API is a static API that the catalog-api command line tool generates from a repository of integrations, such as https://github.com/sensu/catalog. The Sensu web UI uses the generated API files to determine which integrations to display in the Sensu Catalog.

Get the latest catalog SHA-256 checksum

Retrieves the latest catalog content version's SHA-256 checksum. The Sensu web UI uses the checksum information to determine the latest API subpath.

Example

The following example queries the Sensu Catalog API for the latest content version:

```
curl -X GET \
/version.json
```

The request returns the latest catalog content version's SHA-256 checksum and the time of the last update (in seconds since the Unix epoch):

```
{
  "release_sha256":
"af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6",
  "last_updated": 1643664852
}
```

API Specification

/version.json (GET)	
description	Retrieves the latest content version's SHA-256 checksum, which the Sensu web UI uses to determine the latest API subpath. Also returns the time of the last update in seconds since the Unix epoch.
endpoint	/version.json
output	<pre>{ "release_sha256": "af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb 929dcf6", "last_updated": 1643664852 }</pre>
response codes	The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content. ¬ **Error**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get all integration namespaces and names

Retrieves the list of integration namespaces and names for the catalog.

Example

The following example queries the Sensu Catalog API for integration namespaces and names:

```
curl -X GET \
```

```
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/catalog.json
```

The request returns the list of integration namespaces and names:

```
{
  "namespaced_integrations": {
    "nginx": [
      "nginx-monitoring"
    ],
    "system": [
      "host-monitoring"
    ]
  }
}
```

API Specification

/<release_sha256>/v1/catalog.json (GET)	
description	Retrieves the list of integration namespaces and names for the catalog.
endpoint	/<release_sha256>/v1/catalog.json
output	{ "namespaced_integrations": { "nginx": ["nginx-monitoring"], "system": ["host-monitoring"] } }

response codes	The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content. ¬ **Error**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get the configuration and versions for an integration

For the specified integration, retrieves the configuration for the latest version and a list of versions.

Example

The following example queries the Sensu Catalog API for an integration's configuration and versions:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring.json
```

The request returns the latest content version's SHA-256 checksum and the time of the last content update in seconds since the Unix epoch:

```
{
  "metadata": {
    "name": "nginx-monitoring",
    "namespace": "nginx"
  },
  "display_name": "NGINX Monitoring",
  "class": "community",
  "contributors": [
    "@nixwiz",
    "@calebhailey"
  ],
  "provider": "agent/check",
  "short_description": "NGINX monitoring",
```

```
  "supported_platforms": [
    "darwin",
    "linux",
    "windows"
  ],
  "tags": [
    "http",
    "nginx",
    "webserver"
  ],
  "versions": [
    "20220125.0.0",
    "20220126.0.0"
  ]
}
```


API Specification

/<release_sha256>/v1/<namespace>/<name>.json (GET)	
description	Retrieves the specified integration's latest configuration and a list of versions.
endpoint	/<release_sha256>/v1/<namespace>/<name>.json
output	`{ "metadata": { "name": "nginx-monitoring", "namespace": "nginx" }, "display_name": "NGINX Monitoring", "class": "community",`

```
  "contributors": [
    "@nixwiz",
    "@calebhailey"
  ],
  "provider":
"agent/check",
  "short_description":
"NGINX monitoring",

"supported_platforms":
[
    "darwin",
    "linux",
    "windows"
  ],
  "tags": [
    "http",
    "nginx",
    "webserver"
  ],
  "versions": [
    "20220125.0.0",
    "20220126.0.0"
  ]
}
```

---

response codes

The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content.

- **Error**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get all versions for an integration

Retrieves the list of available versions for the specified integration.

Example

The following example queries the Sensu Catalog API for an integration's available versions:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring/versions.json
```

The request returns the integration's available versions:

```
[
  "20220125.0.0",
  "20220126.0.0"
]
```

API Specification

/<release_sha256>/v1/<namespace>/<name>/versions.json (GET)	
description	Retrieves a list of the available versions for the specified integration.
endpoint	/<release_sha256>/v1/<namespace>/<name>/versions.json
output	`[`

```
  "20220125.0.0"
  ,
  "20220126.0.0"
  ]
```

response codes	The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content. - **Error**: 404 (Not Found) - **Error**: 500 (Internal Server Error)

Get the configuration for an integration version

Retrieves the configuration for the specified version of an integration.

Example

The following example queries the Sensu Catalog API for the specified version of an integration:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring/20220125.0.0.json
```

The request returns the configuration for the specified version of the integration:


```
{
  "metadata": {
    "name": "nginx-monitoring",
    "namespace": "nginx"
  },
  "class": "community",
  "contributors": [
    "@nixwiz",
    "@calebhailey"
  ],
  "provider": "agent/check",
  "short_description": "NGINX monitoring",
  "supported_platforms": [
    "darwin",
    "linux",
    "windows"
  ],
  "tags": [
    "http",
    "nginx",
    "webserver"
  ],
  "version": "20220125.0.0"
}
```


API Specification

/<release_sha256>/v1/<namespace>/<name>/<version>.json (GET)	
description	Retrieves the latest content version's SHA-256 checksum, which the Sensu web UI uses to determine the latest API

	subpath.
endpoint	/<release_sha256>/v1/<namespace>/<name>/<version>.json
output	{ "metadata": { "name": "nginx-monitoring", "namespace": "nginx" }, "class": "community", "contributors": ["@nixwiz", "@calebhailey"], "provider": "agent/check", "short_description": "NGINX monitoring", "supported_platforms": [

```
"darwin",
    "linux",

"windows"
  ],
  "tags": [
    "http",
    "nginx",

"webserver"
  ],
  "version":
"20220125.0.
0"
}
```


response codes	The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content. ¬ **Error**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get the Sensu resources for an integration

Retrieves the the Sensu resources for the specified integration version, in JSON format.

***NOTE**: The `/<release_sha256>/v1/<namespace>/<name>/<version>/sensu-resources.json` endpoint does not include assets in the retreived Sensu resources.*

Example

The following example queries the Sensu Catalog API for the Sensu resources for the specified integration version:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring/20220125.0.0/sensu-resources.json
```

The request returns the Sensu resources for the requested integration version:

```
{
  "api_version": "core/v2",
  "metadata": {
    "name": "nginx-healthcheck"
  },
  "spec": {
    "command": "check-nginx-status.rb --url {{ .annotations.check_nginx_status_url | default \"http://localhost:80/nginx_status\" }}",
    "interval": 30,
    "pipelines": [
      {
        "api_version": "core/v2",
        "name": "alerts",
        "type": "Pipeline"
      },
      {
        "api_version": "core/v2",
        "name": "incident-management",
        "type": "Pipeline"
      }
    ],
    "publish": true,
```

```
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-nginx:3.1.2",
      "sensu/sensu-ruby-runtime:0.0.10"
    ],
    "subscriptions": [
      "nginx"
    ],
    "timeout": 10
  },
  "type": "CheckConfig"
}
```

API Specification

/<release_sha256>/v1/<namespace>/<name>/<version>/sensu-resources.json (GET)

description

endpoint

output

response codes

Get the integration README

Retrieves the README for the specified integration version in Markdown format.

Example

The following example queries the Sensu Catalog API for the README for the specified integration version:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring/20220125.0.0/readme.md
```

The request returns the README for the specified integration version in Markdown format.

API Specification

/<release_sha256>/v1/<namespace>/<name>/<version>/readme.md (GET)	
description	Retrieves the README for the specified

	integration version in Markdown format.
endpoint	/<release_sha256>/v1/<namespace>/<name>/<version>/readme.md
output	README in Markdown format
response codes	The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content. ¬ **Error**: 404

(NotFound)

- **Error**: 500 (InternalServerError)

Get the integration changelog

Retrieves the changelog for the specified integration version in Markdown format.

Example

The following example queries the Sensu Catalog API for the changelog for the specified integration

version:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring/20220125.0.0/changelog.md
```

The request returns the changelog for the specified integration version in Markdown format.

API Specification

/<release_sha256>/v1/<namespace>/<name>/<version>/changelog.png (GET)	
description	Retrieves the changelog for the specified integration version in Markdown format
endpoint	/<release_sha256>/v1/<namespace>/<name>/<version>/changelog.md
output	Changelog in Markdown format
response codes	The Catalog

API is statically generated, so response codes indicate an issue with the webserver that is serving the content.

Get the integration logo

Retrieves the logo for the specified integration version in PNG format.

Example

The following example queries the Sensu Catalog API for the logo for the specified integration version:

```
curl -X GET \
/af3c54b86b90fac8977f1bdc80d955002dd3f441bdbb4cc603c94abbb929dcf6/v1/nginx/nginx-monitoring/20220125.0.0/logo.md
```

The request returns the logo for the specified integration version in PNG format.

API Specification

/<release_sha256>/v1/<namespace>/<name>/<version>/logo.png (GET)

description	Retrieves the logo for the specified integration version in PNG format.
endpoint	/<release_sha256>/v1/<namespace>/<name>/<version>/logo.md
output	Logo in PNG format
response codes	The Catalog API is statically generated, so response codes indicate an issue with the webserver that is serving the content. - **Error**: 404 (Not Found) - **Error**: 500 (Internal Serv

er
Error
)

API

API version: v2

The Sensu backend REST API provides a centrally managed control plane for automated, repeatable monitoring and observability workflow configuration and observation event data access.

If you have a healthy clustered backend, you only need to make Sensu API calls to any one of the cluster members. The cluster protocol will replicate your changes to all cluster members.

For information about the Sensu agent API, read the agent reference.

Available APIs

Access all of the data and functionality of Sensu's first-class API clients, sensuctl and the web UI, with Sensu's backend REST APIs. Use the Sensu APIs and endpoints to customize your workflows and integrate your favorite Sensu features with other tools and products.

core/v2 API endpoints

The core/v2 API includes endpoints for the following Sensu resources:

- core/v2/apikeys
- core/v2/assets
- core/v2/checks
- core/v2/cluster
- core/v2/clusterrolebindings
- core/v2/clusterroles
- core/v2/entities
- core/v2/events
- core/v2/filters
-

- core/v2/handlers
- core/v2/hooks
- core/v2/mutators
- core/v2/namespaces
- core/v2/pipelines
- core/v2/rolebindings
- core/v2/roles
- core/v2/silenced
- core/v2/tessen
- core/v2/users

Enterprise APIs

***COMMERCIAL FEATURE**: Access Sensu's enterprise APIs in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

The enterprise APIs include:

- enterprise/authentication/v2
- enterprise/bsm/v1
- enterprise/federation/v1
- enterprise/pipeline/v1
- enterprise/prune/v1alpha
- enterprise/searches/v1
- enterprise/secrets/v1
- enterprise/store/v1
- enterprise/web/v1

Other endpoints

Sensu offers additional endpoints for basic authentication, health, license, metrics, and version:

- /auth
- /health
- /license
- /metrics
- /ready
- /version

URL format

Most core/v2 API endpoints use the standard URL format `/api/core/<version>/namespaces/<namespace>` where:

- `<version>` is the API version: `v2`.
- `<namespace>` is the namespace name.

For enterprise APIs for namespaced resources, the URL format also includes a group that indicates the relevant enterprise feature: `/api/enterprise/<group>/<version>/namespaces/<namespace>`.

For enterprise APIs for cluster-wide resources, the URL format does not include namespace elements: `/api/enterprise/<group>/<version>`.

The endpoint-only APIs do not follow a standard URL format.

Namespaces in API URLs

The examples in the API documentation use the `default` namespace.

The Sensu API requires the authenticated user to have the correct access permissions for the namespace specified in the URL. If the authenticated user has the correct cluster-wide permissions, you can leave out the `/namespaces/<namespace>` portion of the URL to access Sensu resources across namespaces.

Read the RBAC reference for more information about configuring Sensu users and access controls.

Data format

The Sensu API uses JSON-formatted requests and responses.

In terms of output formats, the Sensu API uses `json` output format for responses for APIs in the `core` group. For APIs that are not in the `core` group, responses are in the `wrapped-json` output format. The `wrapped-json` format includes an outer-level `spec` "wrapping" for resource attributes and lists the resource `type` and `api_version`.

Sensu sends events to the backend in `json` format, without the `spec` attribute wrapper or `type` and `api_version` attributes.

Versioning

The Sensu Go API is versioned according to the format `v{majorVersion}{stabilityLevel}{iterationNumber}`, in which `v2` is stable version 2. The Sensu API guarantees backward compatibility for stable versions of the API.

Sensu does not guarantee that an alpha or beta API will be maintained for any period of time. Consider alpha versions under active development — they may not be published for every release. Beta APIs are more stable than alpha versions, but they offer similarly short-lived lifespans and also are not guaranteed to convert programmatically when the API is updated.

Request size limit

The default limit for API request body size is 0.512 MB. Use the `api-request-limit` backend configuration option to customize the API request body size limit if needed.

Access control

With the exception of the authentication, health, metrics, ready, and version API endpoints, the Sensu API requires authentication using a JSON Web Token (JWT) access token or API key.

Code examples in the Sensu API docs use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.

***NOTE**: The authentication information on this page is specific to the Sensu API. For information about using Sensu's built-in basic authentication or external authentication providers to authenticate to the Sensu web UI, API, or sensuctl, read the Control Access documentation.*

Authentication quickstart

To set up a local API testing environment, save your Sensu credentials and access token as environment variables.

Save your Sensu credentials as environment variables:

```
export SENSU_USER=YOUR_USERNAME && SENSU_PASS=YOUR_PASSWORD
```

Save your Sensu access token as an environment variable:

***NOTE**: The command to save your access token as an environment variable requires curl and jq.*

```
export SENSU_ACCESS_TOKEN=`curl -X GET -u "$SENSU_USER:$SENSU_PASS" -s
http://localhost:8080/auth | jq -r ".access_token"`
```

The sensuctl reference demonstrates how to use the `sensuctl env` command to export your access token, token expiry time, and refresh token as environment variables.

Authenticate with /auth API endpoints

Use the authentication API and your Sensu username and password to generate access tokens and refresh tokens. The `/auth` API endpoint lets you generate short-lived API tokens using your Sensu username and password.

1. Retrieve an access token for your user. For example, to generate an access token using example admin credentials:

   ```
   curl -u 'YOUR_USERNAME:YOUR_PASSWORD' http://localhost:8080/auth
   ```

The access token should be included in the output, along with a refresh token:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1544582187,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
```

The access and refresh tokens are JWTs that Sensu uses to digitally sign the details of users' authenticated Sensu sessions.

2. Use the access token in the authentication header of the API request. For example:

```
curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events
```

3. Refresh your access token every 15 minutes. Access tokens last for approximately 15 minutes. When your token expires, you should receive a `401 Unauthorized` response from the API. To generate a new access token, use the `/auth/token` API endpoint, including the expired access token in the authorization header and the refresh token in the request body:

```
curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
-H 'Content-Type: application/json' \
-d '{"refresh_token": "eyJhbGciOiJIUzI1NiIs..."}' \
http://127.0.0.1:8080/auth/token
```

The new access token should be included in the output:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1561055277,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
```

Generate an API access token with sensuctl

You can also generate an API access token using the sensuctl command line tool. The user credentials that you use to configure sensuctl determine your permissions to get, list, create, update, and delete resources with the Sensu API.

1. Install and configure sensuctl.

2. Retrieve an access token for your user:

   ```
   cat ~/.config/sensu/sensuctl/cluster|grep access_token
   ```

   The access token should be included in the output:

   ```
   "access_token": "eyJhbGciOiJIUzI1NiIs...",
   ```

3. Copy the access token into the authentication header of the API request. For example:

   ```
   curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
   http://127.0.0.1:8080/api/core/v2/namespaces/default/events
   ```

4. Refresh your access token every 15 minutes. Access tokens last for approximately 15 minutes. When your token expires, you should receive a `401 Unauthorized` response from the API. To regenerate a valid access token, run any sensuctl command (like `sensuctl event list`) and repeat step 2.

Authenticate with an API key

Each Sensu API key (core/v2/apikey) is a persistent universally unique identifier (UUID) that maps to a stored Sensu username. The advantages of authenticating with API keys rather than access tokens include:

- **More efficient integration**: Check and handler plugins and other code can integrate with the Sensu API without implementing the logic required to authenticate via the `/auth` API endpoint to periodically refresh the access token
- **Improved security**: API keys do not require providing a username and password in check or

handler definitions

- **Better admin control**: API keys can be created and revoked without changing the underlying user's password, but keep in mind that API keys will continue to work even if the user's password changes

API keys are cluster-wide resources, so only cluster admins can grant, view, and revoke them.

> ***NOTE**: API keys are not supported for authentication providers such as LDAP and OIDC.*

Configure an environment variable for API key authentication

Configure the `SENSU_API_KEY` environment variable with your own API key to use it for authentication in your Sensu API requests as shown in the Sensu API code examples.

Follow these steps to generate an API key and export it to the `SENSU_API_KEY` environment variable:

1. Generate an API key with sensuctl:

   ```
   sensuctl api-key grant admin
   ```

   The response will include the new API key:

   ```
   Created: /api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
   ```

   > ***PRO TIP**: Sensuctl is the most direct way to generate an API key, but you can also use the POST core/v2/apikeys endpoint.*

2. Export your API key to the `SENSU_API_KEY` environment variable:

   BASH

   ```
   export SENSU_API_KEY="83abef1e-e7d7-4beb-91fc-79ad90084d5b"
   ```

   CMD

```
SET SENSU_API_KEY="83abef1e-e7d7-4beb-91fc-79ad90084d5b"
```

POWERSHELL

```
$Env:SENSU_API_KEY = "83abef1e-e7d7-4beb-91fc-79ad90084d5b"
```

Authorization header for API key authentication

Similar to the `Bearer [token]` Authorization header, `Key [api-key]` will be accepted as an Authorization header for authentication.

For example, a JWT `Bearer [token]` Authorization header might be:

```
curl -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

If you're using `Key [api-key]` to authenticate instead, the Authorization header might be:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

Example

This example uses the API key directly (rather than the `$SENSU_API_KEY` environment variable) to authenticate to core/v2/checks:

```
curl -H "Authorization: Key 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK` and the definitions for the checks in the default namespace.

Pagination

The Sensu API supports response pagination for most core/v2 GET endpoints that return an array. You can request a paginated response with the `limit` and `continue` query parameters.

Limit query parameter

The following request limits the response to a maximum of two objects:

```
curl http://127.0.0.1:8080/api/core/v2/users?limit=2 -H "Authorization: Key $SENSU_API_KEY"
```

The response includes the available objects up to the specified limit.

Continue query parameter

If more objects are available beyond the limit you specified in a request, the response header includes a `Sensu-Continue` token you can use to request the next page of objects.

For example, the following response indicates that more than two users are available because it provides a `Sensu-Continue` token in the response header:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Continue: L2RlZmF1bU2Vuc3UtTWFjQ
Sensu-Entity-Count: 3
Sensu-Entity-Limit: 100
Sensu-Entity-Warning:
Date: Fri, 14 Feb 2020 15:44:25 GMT
Content-Length: 132
[
  {
    "username": "alice",
    "groups": [
      "ops"
    ],
```

```
    "disabled": false
  },
  {
    "username": "bob",
    "groups": [
      "ops"
    ],
    "disabled": false
  }
]
```

To request the next two available users, use the `Sensu-Continue` token included in the response header:

```
curl http://127.0.0.1:8080/api/core/v2/users?limit=2&continue=L2RlZmF1bU2Vuc3UtTWFjQ
\
-H "Authorization: Key $SENSU_API_KEY"
```

If the response header does not include a `Sensu-Continue` token, there are no further objects to return. For example, this response header indicates that no further users are available:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Entity-Count: 3
Sensu-Entity-Limit: 100
Sensu-Entity-Warning:
Date: Fri, 14 Feb 2020 15:46:02 GMT
Content-Length: 54
[
  {
    "username": "alice",
    "groups": [
      "ops"
    ],
    "disabled": false
  }
]
```

Etag response headers

All GET and PATCH requests return an Etag HTTP response header that identifies a specific version of the resource.Use the Etag value from the response header to conditionally execute PATCH requests that use the If-Match and If-None-Match headers.

If Sensu cannot execute a PATCH request because one of the conditions failed, the request will return the HTTP response code `412 Precondition Failed`.

If-Match example

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-H 'If-Match: "drrn157624731797"' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK`.

If-None-Match example

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-H 'If-None-Match: "drrn157624731797", "reew237527931897"' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
```

```
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK`.

Response filtering

***COMMERCIAL FEATURE**: Access API response filtering in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu API supports response filtering for all GET endpoints that return an array. You can filter resources based on their labels with the `labelSelector` query parameter and based on certain pre-determined fields with the `fieldSelector` query parameter.

***NOTE**: To search based on fields and labels in the Sensu web UI, read Search in the web UI.*

Label selector

The `labelSelector` query parameter allows you to group resources by the label attributes specified in the resource metadata object. All resources support labels within the metadata object.

Field selector

The `fieldSelector` query parameter allows you to organize and select subsets of resources based on certain fields. Here's the list of available fields:

Resource	Fields
Asset	`asset.name` `asset.namespace` `asset.filters`
Check	`check.name` `check.namespace` `check.handlers` `check.publish` `check.round_robin` `check.runtime_assets` `check.subscriptions`

ClusterRole	`clusterrole.name`
ClusterRoleBinding	`clusterrolebinding.name` `clusterrolebinding.role_ref.name` `clusterrolebinding.role_ref.type`
Entity	`entity.name` `entity.namespace` `entity.deregister` `entity.entity_class` `entity.subscriptions`
Event	`event.name` `event.namespace` `event.is_silenced` `event.check.handlers` `event.check.is_silenced` `event.check.name` `event.check.publish` `event.check.round_robin` `event.check.runtime_assets` `event.check.state` `event.check.status` `event.check.subscriptions` `event.entity.deregister` `event.entity.entity_class` `event.entity.name` `event.entity.subscriptions`
Extension	`extension.name` `extension.namespace`
Filter	`filter.name` `filter.namespace` `filter.action` `filter.runtime_assets`
Handler	`handler.name` `handler.namespace` `handler.filters` `handler.handlers` `handler.mutator` `handler.type`
Hook	`hook.name` `hook.namespace`
Mutator	`mutator.name` `mutator.namespace` `mutator.runtime_assets`
Namespace	`namespace.name`
Pipeline	`pipeline.name` `pipeline.namespace`
Role	`role.name` `role.namespace`
RoleBinding	`rolebinding.name` `rolebinding.namespace` `rolebinding.role_ref.name` `rolebinding.role_ref.type`
Secrets	`secret.name` `secret.namespace` `secret.provider` `secret.id`
SecretsProviders	`provider.name`
Silenced	`silenced.name` `silenced.namespace` `silenced.check` `silenced.creator` `silenced.expire_on_resolve` `silenced.subscription`

User	`user.username` `user.disabled` `user.groups`

API-specific syntax

To create an API response filter, you'll write a brief filter statement. The operators and examples sections demonstrate how to construct API response filter statements for different operators and specific purposes.

The filter statement construction is slightly different for different operators, but there are a few general syntax rules that apply to all filter statements.

Spaces in the filter statement

As shown in this example:

```
'fieldSelector=silenced.expire_on_resolve == true'
```

- **Do not** use spaces around the `=` between the selector type and the rest of the filter statement.
- **Do** use spaces around the operator (in this example, the `==`).

Quotation marks around the filter statement

Place the entire filter statement inside single quotes:

```
'fieldSelector=linux in check.subscriptions'
```

Exception: If the filter statement contains a *shell* variable, you must use double quotation marks around the statement:

```
"labelSelector=host == $HOSTNAME"
```

If you use single quotes around a filter statement that contains a shell variable, the single quotes will keep the variable intact instead of expanding it.

> ***NOTE**: This exception only applies to shell variables. It does not apply for variables in languages that treat single and double quotation marks interchangeably, like JavaScript.*

Values that begin with a number or include special characters

If you are filtering for a value that begins with a number, place the value in double quotes:

```
'fieldSelector=entity.name == "1b04994n"'
```

Likewise, to use a label or field selector with string values that include special characters like hyphens and underscores, place the value in double quotes:

```
'labelSelector:region == "us-west-1"'
```

Filter operators

Sensu's API response filtering supports two equality-based operators, two set-based operators, one substring matching operator, and one logical operator.

operator	description	example
`==`	Equality	`check.publish == true`
`!=`	Inequality	`check.namespace != "default"`
`in`	Included in	`linux in check.subscriptions`
`notin`	Not included in	`slack notin check.handlers`
`matches`	Substring matching	`check.name matches "linux-"`

`&&`	Logical AND	`check.publish == true && slack in check.handlers`

Equality-based operators

Sensu's two *equality-based* operators are `==` (equality) and `!=` (inequality).

For example, to retrieve only checks with the label `type` and value `server`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'labelSelector=type == "server"'
```

***NOTE**: Use the flag `--data-urlencode` in cURL to encode the query parameter. Include the `-G` flag so the request appends the query parameter data to the URL.*

To retrieve checks that are not in the `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.namespace != "production"'
```

Set-based operators

Sensu's two *set-based* operators for lists of values are `in` and `notin`.

For example, to retrieve checks with a `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=linux in check.subscriptions'
```

To retrieve checks that do not use the `slack` handler:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=slack notin check.handlers'
```

The `in` and `notin` operators have two important conditions:

- First, they only work when the underlying value you're filtering for is a string. You can filter for strings and arrays of strings with `in` and `notin` operators, but you cannot use them to filter for integer, float, array, or Boolean values.
- Second, to filter for a string, the string must be to the **left** of the operator: `string [in|notin] selector`. To filter for an array of strings, the array must be to the **right** of the operator: `selector [in|notin] [string1,string2]`.

Substring matching operator

Sensu's *substring matching* operator is `matches`.

For example, to retrieve all checks whose name includes `linux`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.name matches "linux"'
```

Suppose you are using Sensu to monitor 1000 entities that are named incrementally and according to technology. For example, your webservers are named `webserver-1` through `webserver-25`, and your CPU entities are named `cpu-1` through `cpu-300`, and so on. In this case, you can use `matches` to retrieve all of your `webserver` entities:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector=entity.name matches "webserver-"'
```

Similarly, if you have entities labeled for different regions, you can use `matches` to find the entities that are labeled for the US (for example, `us-east-1`, `us-west-1`, and so on):

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'labelSelector:region matches "us"'
```

The `matches` operator only works when the underlying value you're filtering for is a string. You can filter for strings and arrays of strings with the `matches` operator, but you cannot use it to filter for integer, float, array, or Boolean values. Also, the string must be to the **right** of the operator: `selector matches string`.

Logical operator

Sensu's logical operator is `&&` (AND). Use it to combine multiple statements separated with the logical operator in field and label selectors.

For example, the following cURL request retrieves checks that are not configured to be published **and** include the `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.publish != true && linux in
check.subscriptions'
```

To retrieve checks that are not published, include a `linux` subscription, and are in the `dev` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.publish != true && linux in check.subscriptions
&& dev in check.namespace'
```

***NOTE**: Sensu does not have the `OR` logical operator.*

Combined selectors

You can use field and label selectors in a single request. For example, to retrieve only checks that include a `linux` subscription *and* do not include a label for type `server`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=linux in check.subscriptions' \
--data-urlencode 'labelSelector=type != "server"'
```

Examples

Values with special characters

To use a label or field selector with string values that include special characters like hyphens and underscores, place the value in single or double quotes:

```
curl -H "Authorization: Key $SENSU_API_KEY" -X GET
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'labelSelector=region == "us-west-1"'
```

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector="entity:i-0c1f8a116b84ea50c" in entity.subscriptions'
```

Use selectors with arrays of strings

To retrieve checks that are in either the `dev` or `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.namespace in [dev,production]'
```

Filter events by entity or check

To retrieve events for a specific check (`checkhttp`):

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=checkhttp in event.check.name'
```

Similary, to retrieve only events for the `server` entity:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=server in event.entity.name'
```

Filter events by severity

Use the `event.check.status` field selector to retrieve events by severity.For example, to retrieve all events at `2` (CRITICAL) status:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=event.check.status == "2"'
```

Filter all incidents

To retrieve all incidents (all events whose status is not `0`):

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=event.entity.status != "0"'
```

Filter checks, entities, or events by subscription

To list all checks that include the `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=linux in check.subscriptions'
```

Similarly, to list all entities that include the `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector=linux in entity.subscriptions'
```

To list all events for the `linux` subscription, use the `event.entity.subscriptions` field selector:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=linux in event.entity.subscriptions'
```

Filter silenced resources and silences

Filter silenced resources by namespace

To list all silenced resources for a particular namespace (in this example, the `default` namespace):

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.namespace == "default"'
```

Likewise, to list all silenced resources *except* those in the `default` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.namespace != "default"'
```

To list all silenced events for all namespaces:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=event.is_silenced == true'
```

Filter silences by creator

To list all silences created by the user `alice`:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.creator == "alice"'
```

To list all silences that were not created by the `admin` user:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.creator != "admin"'
```

Filter silences by silence subscription

To retrieve silences with a specific subscription (in this example, `linux`):

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.subscription == "linux"'
```

Another way to make the same request is:

```
curl -H "Authorization: Key $SENSU_API_KEY"
```

```
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=linux in silenced.subscription'
```

***NOTE**: For this field selector, `subscription` means the subscription specified for the silence. In other words, this filter retrieves **silences** with a particular subscription, not silenced entities or checks with a matching subscription.*

Filter silenced resources by expiration

To list all silenced resources that expire only when a matching check resolves:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.expire_on_resolve == true'
```

Core API

Sensu's core/v2 API provides GET, POST, PUT, and DELETE access to Sensu events and resources. The core/v2 API includes endpoints for the following Sensu resources:

- core/v2/apikeys
- core/v2/assets
- core/v2/checks
- core/v2/cluster
- core/v2/clusterrolebindings
- core/v2/clusterroles
- core/v2/entities
- core/v2/events
- core/v2/filters
- core/v2/handlers
- core/v2/hooks
- core/v2/mutators
- core/v2/namespaces
- core/v2/pipelines
- core/v2/rolebindings
- core/v2/roles
- core/v2/silenced
- core/v2/tessen
- core/v2/users

core/v2/apikeys

***NOTE**: Requests to `core/v2/apikeys` endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all API keys

The `/apikeys` GET endpoint retrieves all API keys.

Example

The following example demonstrates a GET request to the `/apikeys` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/apikeys \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains all API keys, similar to this example:

```
[
  {
    "metadata": {
      "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b",
      "created_by": "admin"
    },
    "username": "admin",
    "created_at": 1570640363
  },
  {
    "metadata": {
```

```
      "name": "94jhid83j-96kg-2ewr-bab3-ppd3d49tdd94",
      "created_by": "admin"
    },
    "username": "admin",
    "created_at": 1651257929
  }
]
```


API Specification

/apikeys (GET)	
description	Returns the list of API keys.
example url	http://hostname:8080/api/core/v2/apikeys
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters. Read the API overview for details.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "metadata": { "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b", "created_by": "admin" }, "username": "admin", "created_at": 1570640363 }, { "metadata": { "name": "94jhid83j-96kg-2ewr-bab3-ppd3d49tdd94", "created_by": "admin"`

```
    },
    "username": "admin",
    "created_at": 1651257929
  }
]
```

Create a new API key

The `/apikeys` API endpoint provides HTTP POST access to create a new API key.

Example

In the following example, an HTTP POST request is submitted to the `/apikeys` API endpoint to create a new API key.

***NOTE**: For the `/apikeys` POST endpoint, authenticate with a Sensu access token, which you can generate with /auth API endpoints or sensuctl. This example uses `$SENSU_ACCESS_TOKEN` to represent a valid Sensu access token.*

If you prefer, you can create a new API key with sensuctl instead of using this endpoint.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "username": "admin"
}' \
http://127.0.0.1:8080/api/core/v2/apikeys
```

The request returns a successful HTTP `HTTP/1.1 201 Created` response, along with a `Location` header that contains the relative path to the new API key.

API Specification

/apikeys (POST)	
description	Creates a new API key, a Sensu-generated universally unique identifier (UUID). The response will include HTTP 201 and a `Location` header that contains the relative path to the new API key.
example URL	http://hostname:8080/api/core/v2/apikeys
request payload	{ "username": "admin" }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific API key

The `/apikeys/:apikey` GET endpoint retrieves the specified API key.

Example

The following example queries the `/apikeys/:apikey` API:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b \
-H "Authorization: Key $SENSU_API_KEY"
```

The request returns a successful `HTTP/1.1 200 OK` response and the requested `:apikey` definition, similar to the example below, or an error if the key is not found:

```
{
  "metadata": {
    "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b",
    "created_by": "admin"
  },
  "username": "admin",
  "created_at": 1570640363
}
```


API Specification

/apikeys/:apikey (GET)	
description	Returns the specified API key.
example url	http://hostname:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	`{ "metadata": { "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b", "created_by": "admin" }, "username": "admin", "created_at": 1570640363 }`

Update an API key with PATCH

The `/apikeys/:apikey` PATCH endpoint updates the specified API key.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.*

Example

The following example queries the `/apikeys/:apikey` API updates the username for the specified `:apikey` definition and returns a successful `HTTP/1.1 200 OK` response.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
{
  "username": "devteam"
} \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
```

API Specification

/apikeys/:apikey (PATCH)	
description	Updates the specified API key.
example url	http://hostname:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found)

	¬ **Error**: 500 (Internal Server Error)
output	<pre>{ "username": "devteam" }</pre>

Delete an API key

The `/apikeys/:apikey` API endpoint provides HTTP DELETE access to remove an API key.

Example

The following example shows a request to the `/apikeys/:apikey` API endpoint to delete the API key `83abef1e-e7d7-4beb-91fc-79ad90084d5b`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
```

API Specification

/apikeys/:apikey (DELETE)	
description	Revokes the specified API key.
example URL	http://hostname:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
response codes	¬ **Success**: 204 (No Content) ¬ **Error**: 500 (Internal Server Error)

core/v2/assets

***NOTE**: Requests to `core/v2/assets` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all assets

The `/assets` API endpoint provides HTTP GET access to dynamic runtime asset data.

Example

The following example demonstrates a GET request to the `/assets` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains dynamic runtime asset definitions, similar to this example:

```
[
  {
    "url": "https://github.com/sensu/sensu-influxdb-
handler/releases/download/3.1.2/sensu-influxdb-handler_3.1.2_linux_amd64.tar.gz",
    "sha512":
"612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9b
1257bb479676bc155b24e21bf93c722b812b0f15cb3bd",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
```

```
    "builds": null,
    "metadata": {
      "name": "sensu-influxdb-handler",
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  },
  {
    "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
    "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

API Specification

/assets (GET)	
description	Returns the list of dynamic runtime assets.

example url	http://hostname:8080/api/core/v2/namespaces/default/assets
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "url": "https://github.com/sensu/sensu-influxdb-handler/releases/download/3.1.2/sensu-influxdb-handler_3.1.2_linux_amd64.tar.gz", "sha512": "612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9b1257bb479676bc155b24e21bf93c722b812b0f15cb3bd", "filters": ["entity.system.os == 'linux'", "entity.system.arch == 'amd64'"], "builds": null, "metadata": { "name": "sensu-influxdb-handler", "namespace": "default", "created_by": "admin" }, "headers": { "Authorization": "Bearer $TOKEN", "X-Forwarded-For": "client1, proxy1, proxy2" } }, { "url": "https://github.com/sensu/sensu-slack-handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",

```
    "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66
556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eedb706
4df9019adc8",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

Create a new dynamic runtime asset

The `/assets` API endpoint provides HTTP POST access to dynamic runtime asset data.

Example

In the following example, an HTTP POST request is submitted to the `/assets` API endpoint to create a role named `sensu-slack-handler`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
```

```
  "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
  "filters": [
    "entity.system.os == 'linux'",
    "entity.system.arch == 'amd64'"
  ],
  "headers": {
    "Authorization": "Bearer $TOKEN",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  },
  "metadata": {
    "name": "sensu-slack-handler",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets
```

The request returns a successful `HTTP/1.1 201 Created` response.

API Specification

/assets (POST)	
description	Creates a Sensu dynamic runtime asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets
payload	{ "url": "https://github.com/sensu/sensu-slack-handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz", "sha512": "68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eedb7064df9019adc8", "filters": ["entity.system.os == 'linux'", "entity.system.arch == 'amd64'"

```
  ],
  "headers": {
    "Authorization": "Bearer $TOKEN",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  },
  "metadata": {
    "name": "sensu-slack-handler",
    "namespace": "default"
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific dynamic runtime asset

The `/assets/:asset` API endpoint provides HTTP GET access to dynamic runtime asset data for specific `:asset` definitions, by asset `name`.

Example

The following example queries the `/assets/:asset` API endpoint for the `:asset` named `check_script`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:asset` definition (in this example, for the `:asset` named `check_script`):

```
```

```
[
  {
    "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
    "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```


API Specification

/assets/:asset (GET)	
description	Returns the specified dynamic runtime asset.
example url	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output


```
[
  {
    "url": "https://github.com/sensu/sensu-slack-handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
    "sha512": "68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
    "filters": [
      "entity.system.os = 'linux'",
      "entity.system.arch = 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```


Create or update a dynamic runtime asset

The `/assets/:asset` API endpoint provides HTTP PUT access to create or update specific `:asset` definitions, by dynamic runtime asset name.

Example

In the following example, an HTTP PUT request is submitted to the `/assets/:asset` API endpoint to create the dynamic runtime asset `sensu-slack-handler`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
  "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
  "filters": [
    "entity.system.os == 'linux'",
    "entity.system.arch == 'amd64'"
  ],
  "headers": {
    "Authorization": "Bearer $TOKEN",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  },
  "metadata": {
    "name": "sensu-slack-handler",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/sensu-slack-
handler
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/assets/:asset (PUT)	
description	Creates or updates the specified Sensu dynamic runtime asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
payload	{ "url": "https://github.com/sensu/sensu-slack-

```
handler/releases/download/1.0.3/sensu-slack-
handler_1.0.3_linux_amd64.tar.gz",
  "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a
66556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eed
b7064df9019adc8",
  "filters": [
    "entity.system.os == 'linux'",
    "entity.system.arch == 'amd64'"
  ],
  "headers": {
    "Authorization": "Bearer $TOKEN",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  },
  "metadata": {
    "name": "sensu-slack-handler",
    "namespace": "default"
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a dynamic runtime asset with PATCH

The `/assets/:asset` API endpoint provides HTTP PATCH access to update `:asset` definitions, specified by asset name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/assets/:asset` API endpoint to add a label for the `sensu-slack-handler` asset.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
```

The request will return a successful `HTTP/1.1 200 OK` response.

API Specification

/assets/:asset (PATCH)	
description	Updates the specified Sensu asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
payload	<pre>{ "metadata": { "labels": { "region": "us-west-1" } }</pre>

```
}
```

response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a dynamic runtime asset

The `/assets/:asset` API endpoint provides HTTP DELETE access so you can delete a dynamic runtime assets.

> ***NOTE**: Deleting a dynamic runtime asset does not remove the downloaded files from the asset cache or remove any references to the deleted asset in other resources.*

Example

The following example shows a request to the `/assets/:asset` API endpoint to delete the asset `sensu-slack-handler`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/assets/:asset (DELETE)	
description	Deletes the specified Sensu dynamic runtime asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler

response codes

- **Success**: 204 (No Content)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get a subset of assets with response filtering

The `/assets` API endpoint supports response filtering for a subset of asset data based on labels and the following fields:

- `asset.name`
- `asset.namespace`
- `asset.filters`

Example

The following example demonstrates a request to the `/assets` API endpoint with response filtering that excludes dynamic runtime asset definitions that are in the `production` namespace:

```
curl -H "Authorization: Key X" http://127.0.0.1:8080/api/core/v2/assets -G \
--data-urlencode 'fieldSelector=asset.namespace != "production"'
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only dynamic runtime asset definitions that are **not** in the `production` namespace:

```
[
  {
    "filters": null,
```

```
    "builds": [
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_windows_amd64.tar.gz",
        "sha512":
"900cfdf28d6088b929c4bf9a121b628971edee5fa5cbc91a6bc1df3bd9a7f8adb1fcfb7b1ad70589ed5
b4f5ec87d9a9a3ba95bcf2acda56b0901406f14f69fe7",
        "filters": [
          "entity.system.os == 'windows'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_darwin_amd64.tar.gz",
        "sha512":
"db81ee70426114e4cd4b3f180f2b0b1e15b4bffc09d7f2b41a571be2422f4399af3fbd2fa2918b88319
09ab4bc2d3f58d0aa0d7b197d3a218b2391bb5c1f6913",
        "filters": [
          "entity.system.os == 'darwin'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_armv7.tar.gz",
        "sha512":
"400aacce297176e69f3a88b0aab0ddfdbe9dd6a37a673cb1774c8d4750a91cf7713a881eef26ea21d20
0f74cb20818161c773490139e6a6acb92cbd06dee994c",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'armv7'"
        ],
        "headers": null
      },
      {
        "url":
```

```
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_linux_arm64.tar.gz",
        "sha512": "bef7802b121ac2a2a5c5ad169d6003f57d8b4f5e83eae998a0e0dd1e7b89678d4a62e678d153edacdd65fd1d0123b5f51308622690455e77cec6deccfa183397",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'arm64'"
        ],
        "headers": null
      },
      {
        "url": "https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_linux_386.tar.gz",
        "sha512": "a2dcb5324952567a61d76a2e331c1c16df69ef0e0b9899515dad8d1531b204076ad0c008f59fc2f4735a5a779afb0c1baa132268c41942b203444e377fe8c8e5",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == '386'"
        ],
        "headers": null
      },
      {
        "url": "https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_linux_amd64.tar.gz",
        "sha512": "24539739b5eb19bbab6eda151d0bcc63a0825afdfef3bc1ec3670c7b0a00fbbb2fd006d605a7a038b32269a22026d8947324f2bc0acdf35e8563cf4cb8660d7f",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null
      }
    ],
    "metadata": {
      "name": "check-cpu-usage",
      "namespace": "default",
      "annotations": {
```

```
        "io.sensu.bonsai.api_url":
"https://bonsai.sensu.io/api/v1/assets/sensu/check-cpu-usage",
        "io.sensu.bonsai.name": "check-cpu-usage",
        "io.sensu.bonsai.namespace": "sensu",
        "io.sensu.bonsai.tags": "",
        "io.sensu.bonsai.tier": "Community",
        "io.sensu.bonsai.url": "https://bonsai.sensu.io/assets/sensu/check-cpu-
usage",
        "io.sensu.bonsai.version": "0.2.2"
      },
      "created_by": "admin"
    },
    "headers": null
  }
]
```


API Specification

/assets (GET) with response filters	
description	Returns the list of assets that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/assets
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "filters": null, "builds": [`

```
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e8819
89c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_windows_amd64.tar.gz",
        "sha512":
"900cfdf28d6088b929c4bf9a121b628971edee5fa5c
bc91a6bc1df3bd9a7f8adb1fcfb7b1ad70589ed5b4f5
ec87d9a9a3ba95bcf2acda56b0901406f14f69fe7",
        "filters": [
          "entity.system.os == 'windows'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e8819
89c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_darwin_amd64.tar.gz",
        "sha512":
"db81ee70426114e4cd4b3f180f2b0b1e15b4bffc09d
7f2b41a571be2422f4399af3fbd2fa2918b8831909ab
4bc2d3f58d0aa0d7b197d3a218b2391bb5c1f6913",
        "filters": [
          "entity.system.os == 'darwin'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e8819
89c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_armv7.tar.gz",
        "sha512":
"400aacce297176e69f3a88b0aab0ddfdbe9dd6a37a6
73cb1774c8d4750a91cf7713a881eef26ea21d200f74
cb20818161c773490139e6a6acb92cbd06dee994c",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'armv7'"
```

```
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_linux_arm64.tar.gz",
        "sha512":
"bef7802b121ac2a2a5c5ad169d6003f57d8b4f5e83eae998a0e0dd1e7b89678d4a62e678d153edacdd65fd1d0123b5f51308622690455e77cec6deccfa183397",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'arm64'"
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_linux_386.tar.gz",
        "sha512":
"a2dcb5324952567a61d76a2e331c1c16df69ef0e0b9899515dad8d1531b204076ad0c008f59fc2f4735a5a779afb0c1baa132268c41942b203444e377fe8c8e5",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == '386'"
        ],
        "headers": null
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_linux_amd64.tar.gz",
        "sha512":
"24539739b5eb19bbab6eda151d0bcc63a0825afdfef3bc1ec3670c7b0a00fbbb2fd006d605a7a038b32269a22026d8947324f2bc0acdf35e8563cf4cb8660d7f",
```

```
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null
      }
    ],
    "metadata": {
      "name": "check-cpu-usage",
      "namespace": "default",
      "annotations": {
        "io.sensu.bonsai.api_url":
"https://bonsai.sensu.io/api/v1/assets/sensu
/check-cpu-usage",
        "io.sensu.bonsai.name": "check-cpu-
usage",
        "io.sensu.bonsai.namespace":
"sensu",
        "io.sensu.bonsai.tags": "",
        "io.sensu.bonsai.tier": "Community",
        "io.sensu.bonsai.url":
"https://bonsai.sensu.io/assets/sensu/check-
cpu-usage",
        "io.sensu.bonsai.version": "0.2.2"
      },
      "created_by": "admin"
    },
    "headers": null
  }
]
```

core/v2/checks

***NOTE**: Requests to `core/v2/checks` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all checks

The `/checks` API endpoint provides HTTP GET access to check data.

Example

The following example demonstrates a GET request to the `/checks` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the check definitions in the `default` namespace:

```
[
  {
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
```

```
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default",
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ]
  },
  {
    "command": "http-perf --url http://localhost --warning 1s --critical 2s",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "http-checks"
    ],
    "subscriptions": [
      "webserver"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
```

```
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "nagios_perfdata",
    "output_metric_handlers": "sensu_to_sumo",
    "env_vars": null,
    "metadata": {
      "name": "collect-metrics",
      "namespace": "default",
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": []
  },
  {
    "command": "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
query up",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "sensu-prometheus-collector"
    ],
    "subscriptions": [
      "app_tier"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "influxdb_line",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "prometheus_metrics",
```

```
      "namespace": "default",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": [
      {
        "name": "prometheus_metrics_workflows",
        "type": "Pipeline",
        "api_version": "core/v2"
      }
    ]
  }
]
```

API Specification

/checks (GET)	
description	Returns the list of checks.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { `

```
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default",
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ]
  },
  {
    "command": "http-perf --url http://localhost --warning
1s --critical 2s",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
```

```
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "http-checks"
    ],
    "subscriptions": [
      "webserver"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "nagios_perfdata",
    "output_metric_handlers": "sensu_to_sumo",
    "env_vars": null,
    "metadata": {
      "name": "collect-metrics",
      "namespace": "default",
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": []
  },
  {
    "command": "sensu-prometheus-collector -prom-url
http://localhost:9090 -prom-query up",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "sensu-prometheus-collector"
    ],
    "subscriptions": [
      "app_tier"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
```

```
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "influxdb_line",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "prometheus_metrics",
      "namespace": "default",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": [
      {
        "name": "prometheus_metrics_workflows",
        "type": "Pipeline",
        "api_version": "core/v2"
      }
    ]
  }
]
```

Create a new check

The `/checks` API endpoint provides HTTP POST access to create checks.

Example

In the following example, an HTTP POST request is submitted to the `/checks` API endpoint to create a `check_cpu` check. The request includes the check definition in the request body.

```
curl -X POST \
```

```
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "command": "check-cpu-usage.sh -w 75 -c 90",
  "subscriptions": [
    "system"
  ],
  "interval": 60,
  "publish": true,
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "runtime_assets": [
    "check-cpu-usage"
  ],
  "metadata": {
    "name": "check_cpu",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/checks (POST)	
description	Creates a Sensu check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks
example payload	{ "command": "check-cpu-usage.sh -w 75 -c 90", "subscriptions": [

```
    "system"
  ],
  "interval": 60,
  "publish": true,
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "runtime_assets": [
    "check-cpu-usage"
  ],
  "metadata": {
    "name": "check_cpu",
    "namespace": "default"
  }
}
```

payload parameters	Required check attributes: `interval` (integer) or `cron` (string) and a `metadata` scope that contains `name` (string) and `namespace` (string). For more information about creating checks, read the checks reference.
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific check

The `/checks/:check` API endpoint provides HTTP GET access to `:check` definitions, specified by check name.

Example

The following example queries the `/checks/:check` API endpoint for the `:check` named `check_cpu`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:check` definition (in this example, `check_cpu`):


```
{
  "command": "check-cpu-usage.sh -w 75 -c 90",
  "handlers": [],
  "high_flap_threshold": 0,
  "interval": 60,
  "low_flap_threshold": 0,
  "publish": true,
  "runtime_assets": [
    "check-cpu-usage"
  ],
  "subscriptions": [
    "system"
  ],
  "proxy_entity_name": "",
  "check_hooks": null,
  "stdin": false,
  "subdue": null,
  "ttl": 0,
  "timeout": 0,
  "round_robin": false,
  "output_metric_format": "",
  "output_metric_handlers": null,
  "env_vars": null,
  "metadata": {
    "name": "check_cpu",
    "namespace": "default",
    "created_by": "admin"
  },
  "secrets": null,
```

```
  "pipelines": [
    {
      "name": "incident_alerts",
      "type": "Pipeline",
      "api_version": "core/v2"
    }
  ]
}
```

API Specification

/checks/:check (GET)	
description	Returns the specified check.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "command": "check-cpu-usage.sh -w 75 -c 90", "handlers": [], "high_flap_threshold": 0, "interval": 60, "low_flap_threshold": 0, "publish": true, "runtime_assets": ["check-cpu-usage"], "subscriptions": ["system"],

```
  "proxy_entity_name": "",
  "check_hooks": null,
  "stdin": false,
  "subdue": null,
  "ttl": 0,
  "timeout": 0,
  "round_robin": false,
  "output_metric_format": "",
  "output_metric_handlers": null,
  "env_vars": null,
  "metadata": {
    "name": "check_cpu",
    "namespace": "default",
    "created_by": "admin"
  },
  "secrets": null,
  "pipelines": [
    {
      "name": "incident_alerts",
      "type": "Pipeline",
      "api_version": "core/v2"
    }
  ]
}
```


Create or update a check

The `/checks/:check` API endpoint provides HTTP PUT access to create and update `:check` definitions, specified by check name.

Example

In the following example, an HTTP PUT request is submitted to the `/checks/:check` API endpoint to update the `check_cpu` check:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
```

```
-H 'Content-Type: application/json' \
-d '{
  "command": "check-cpu-usage.sh -w 75 -c 90",
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "interval": 60,
  "publish": true,
  "subscriptions": [
    "system"
  ],
  "metadata": {
    "name": "check_cpu",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/checks/:check (PUT)	
description	Creates or updates the specified Sensu check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu
payload	{ "command": "check-cpu-usage.sh -w 75 -c 90", "pipelines": [{ "api_version": "core/v2", "type": "Pipeline",

```
        "name": "incident_alerts"
      }
    ],
    "interval": 60,
    "publish": true,
    "subscriptions": [
      "system"
    ],
    "metadata": {
      "name": "check_cpu",
      "namespace": "default"
    }
}
```

payload parameters	Required check attributes: `interval` (integer) or `cron` (string) and a `metadata` scope that contains `name` (string) and `namespace` (string). For more information about creating checks, read the checks reference.
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Update a check with PATCH

The `/checks/:check` API endpoint provides HTTP PATCH access to update `:check` definitions, specified by check name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/checks/:check` API endpoint to update the subscriptions array for the `check_cpu` check, resulting in a `HTTP/1.1 200 OK` response and the updated check definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "subscriptions": [
    "system",
    "health"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu
```

API Specification

/checks/:check (PATCH)	
description	Updates the specified Sensu check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu
payload	`{ "subscriptions": [ "system", "health" ] }`
response codes	

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a check

The `/checks/:check` API endpoint provides HTTP DELETE access to delete a check from Sensu, specified by the check name.

Example

The following example shows a request to the `/checks/:check` API endpoint to delete the check named `check_cpu`, which will result in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu
```

API Specification

/checks/:check (DELETE)	
description	Removes the specified check from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu
response codes	- **Success**: 204 (No Content) - **Missing**: 404 (Not Found) - **Error**: 500 (Internal Server Error)

Create an ad hoc check execution request

The `/checks/:check/execute` API endpoint provides HTTP POST access to create an ad hoc check execution request so you can execute a check on demand.

Example

In the following example, an HTTP POST request is submitted to the `/checks/:check/execute` API endpoint to execute the `check_cpu` check. The request includes the check name in the request body.

***PRO TIP**: Include the `subscriptions` attribute with the request body to override the subscriptions configured in the check definition. This gives you the flexibility to execute a check on any Sensu entity or group of entities on demand.*

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "check": "check_cpu",
  "subscriptions": [
    "entity:i-424242"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu/execute
```

The request will return a successful `HTTP/1.1 202 Accepted` response and an `issued` timestamp:

```
{"issued":1543861798}
```

***NOTE**: If you specify a round robin check, Sensu will execute the check on all agents with a matching subscription. After the ad hoc execution, the check will run as scheduled in round robin fashion.*

To execute a round robin check on a single agent, include the agent's entity name subscription in

the request body.For example, if the entity is named `webserver1`, use the subscription `entity:webserver1`.

API Specification

/checks/:check/execute (POST)	
description	Creates an ad hoc request to execute the specified check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu/execute
payload	`{ "check": "check_cpu", "subscriptions": [ "entity:i-424242" ] }`
payload parameters	¬ Required: `check` (the name of the check to execute). ¬ Optional: `subscriptions` (an array of subscriptions to publish the check request to). When provided with the request, the `subscriptions` attribute overrides any subscriptions configured in the check definition.
response codes	¬ **Success**: 202 (Accepted) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Assign a hook to a check

The `/checks/:check/hooks/:type` API endpoint provides HTTP PUT access to assign a hook to a check.

Example

In the following example, an HTTP PUT request is submitted to the `/checks/:check/hooks/:type` API endpoint, assigning the `process_tree` hook to the `check_cpu` check in the event of a `critical` type check result:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "critical": [
    "process_tree"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu/hooks/critical
```

The request returns a successful `HTTP/1.1 201 Created` response.

API Specification

checks/:check/hooks/:type (PUT)	
description	Assigns a hook to a check (specified by the check name and check response type).
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu/hooks/critical
example payload	{ "critical": ["process_tree"]

```
}
```

payload parameters	This endpoint requires a JSON map of check response types (for example, `critical` or `warning`). Each must contain an array of hook names.
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Remove a hook from a check

The `/checks/:check/hooks/:type/hook/:hook` API endpoint provides HTTP DELETE access to a remove a hook from a check.

Example

The following example shows a request to the `/checks/:check/hooks/:type/hook/:hook` API endpoint to remove the `process_tree` hook from the `check_cpu` check, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check_cpu/hooks/critical
/hook/process_tree
```

API Specification

/checks/:check/hooks/:type/hook/:hook (DELETE)	
description	Removes a single hook from a check (specified by the check name, check

	response type, and hook name). Read the checks reference for available types.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check_cpu/hooks/critical/hook/process_tree
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of checks with response filtering

The `/checks` API endpoint supports response filtering for a subset of check data based on labels and the following fields:

- `check.name`
- `check.namespace`
- `check.handlers`
- `check.publish`
- `check.round_robin`
- `check.runtime_assets`
- `check.subscriptions`

Example

The following example demonstrates a request to the `/checks` API endpoint with response filtering for only check definitions whose subscriptions include `system`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
```

```
  --data-urlencode 'fieldSelector="system" in check.subscriptions'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only check definitions whose subscriptions include `system`:


```
[
  {
    "command": "check-cpu-usage.sh -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "subscriptions": [
      "system",
      "health"
    ],
    "proxy_entity_name": "",
    "check_hooks": [
      {
        "critical": [
          "process_tree"
        ]
      }
    ],
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default",
      "created_by": "admin"
```

```
    },
    "secrets": null,
    "pipelines": [
      {
        "name": "incident_alerts",
        "type": "Pipeline",
        "api_version": "core/v2"
      }
    ]
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/checks (GET) with response filters	
description	Returns the list of checks that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/checks
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "command": "check-cpu-usage.sh -w 75 -c 90",`

```
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "subscriptions": [
      "system",
      "health"
    ],
    "proxy_entity_name": "",
    "check_hooks": [
      {
        "critical": [
          "process_tree"
        ]
      }
    ],
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default",
      "created_by": "admin"
    },
    "secrets": null,
    "pipelines": [
      {
        "name": "incident_alerts",
        "type": "Pipeline",
        "api_version": "core/v2"
      }
    ]
  }
```

```
]
```

core/v2/cluster

***NOTE**: Requests to `core/v2/cluster` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all cluster data

The `/cluster/members` API endpoint provides HTTP GET access to Sensu cluster data.

Example

The following example demonstrates a request to the `/cluster/members` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/cluster/members \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains a Sensu cluster definition:

```
{
  "header": {
    "cluster_id": 4255616304056076734,
    "member_id": 9882886658148554927,
    "raft_term": 2
  },
  "members": [
    {
      "ID": 9882886658148554927,
      "name": "default",
      "peerURLs": [
```

```
          "http://127.0.0.1:2380"
        ],
        "clientURLs": [
          "http://127.0.0.1:2379"
        ]
      }
    ]
}
```

API Specification

/cluster/members (GET)	
description	Returns the etcd cluster definition.
example url	http://hostname:8080/api/core/v2/cluster/members
query parameters	`timeout` : Defines the timeout when querying etcd. Default is `3` .
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
example output	{ "header": { "cluster_id": 4255616304056076734, "member_id": 9882886658148554927, "raft_term": 2 }, "members": [{ "ID": 9882886658148554927, "name": "default", "peerURLs": ["http://127.0.0.1:2380"],

```
      "clientURLs": [
        "http://127.0.0.1:2379"
      ]
    }
  ]
}
```

Create a new cluster member

The `/cluster/members` API endpoint provides HTTP POST access to create a Sensu cluster member.

Example

In the following example, an HTTP POST request is submitted to the `/cluster/members` API endpoint to create a Sensu cluster member. The request includes the cluster member peer address in the request URL:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/cluster/members?peer-addrs=http://127.0.0.1:2380
```

The request will return a successful `HTTP/1.1 201 Created` response along with the updated cluster definition:

```
{
  "header": {
    "cluster_id": 4255616304056077000,
    "member_id": 9882886658148555000,
    "raft_term": 2
  },
  "members": [
    {
      "ID": 9882886658148555000,
      "name": "default",
```

```
      "peerURLs": [
        "http://127.0.0.1:2380"
      ],
      "clientURLs": [
        "http://localhost:2379"
      ]
    }
  ]
}
```

API Specification

/cluster/members/:member (POST)	
description	Creates a cluster member.
example url	http://hostname:8080/api/core/v2/cluster/members?peer-addrs=http://127.0.0.1:2380
query parameters	¬ Required: `peer-addrs` (a comma-delimited list of peer addresses).
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Create or update a cluster member

The `/cluster/members/:member` API endpoint provides HTTP PUT access to create or update a cluster member, by the cluster member's hex-encoded ID.

Example

The following example submits an HTTP PUT request to the `/cluster/members/:member` API endpoint to update the member whose hex-encoded ID is `8927110dc66458af`.

> ***IMPORTANT**: The PUT `/cluster/members/:member` URL uses the cluster member's hex-encoded UInt64 ID, **not** the member ID listed in the cluster definition.*
>
> *To get the correct hex-encoded UInt64 ID for the member, run `sensuctl cluster member-list`. The first column in the response lists the ID you need for the PUT `/cluster/members/:member` URL.*

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/cluster/members/8927110dc66458af?peer-
addrs=http://127.0.0.1:2380
```

The request will return a `HTTP/1.1 200 OK` response and the updated cluster definition:

```
{
  "header": {
    "cluster_id": 4255616304056077000,
    "member_id": 9882886658148555000,
    "raft_term": 2
  },
  "members": [
    {
      "ID": 9882886658148555000,
      "name": "default",
      "peerURLs": [
        "http://127.0.0.1:2380"
      ],
      "clientURLs": [
        "http://localhost:2379"
      ]
    }
  ]
}
```

API Specification

/cluster/members/:member (PUT)	
description	Creates or updates a cluster member.
example url	http://hostname:8080/api/core/v2/cluster/members/8927110dc66458af?peer-addrs=http://127.0.0.1:2380
url parameters	Required: Hex-encoded UInt64 cluster member ID generated using `sensuctl cluster member-list` (in this example, `8927110dc66458af`).
query parameters	Required: `peer-addrs` (a comma-delimited list of peer addresses).
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
example output	{ "header": { "cluster_id": 4255616304056077000, "member_id": 9882886658148555000, "raft_term": 2 }, "members": [{ "ID": 9882886658148555000, "name": "default", "peerURLs": ["http://127.0.0.1:2380"], "clientURLs": ["http://localhost:2379"] }]

```
}
```

Delete a cluster member

The `/cluster/members/:member` API endpoint provides HTTP DELETE access to remove a Sensu cluster member.

Example

The following example shows a request to the `/cluster/members/:member` API endpoint to remove the Sensu cluster member with the ID `8927110dc66458af`, which will result in a successful `HTTP/1.1 204 No Content` response.

***IMPORTANT**: The DELETE `/cluster/members/:member` URL uses the cluster member's hex-encoded UInt64 ID, **not** the member ID listed in the cluster definition.*

To get the correct hex-encoded UInt64 ID for the member, run `sensuctl cluster member-list`. The first column in the response lists the ID you need for the DELETE `/cluster/members/:member` URL.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/cluster/members/8927110dc66458af
```

API Specification

/cluster/members/:member (DELETE)	
description	Removes a member from a Sensu cluster (specified by the member ID).
example url	http://hostname:8080/api/core/v2/cluster/members/8927110dc66458af

url parameters	Required: Hex-encoded UInt64 cluster member ID generated using `sensuctl cluster member-list` (in this example, `8927110dc66458af`)
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a cluster ID

The `/cluster/id` API endpoint provides HTTP GET access to the Sensu cluster ID.

Example

The following example demonstrates a request to the `/cluster/id` API endpoint:

```
curl -X GET \
 -H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/cluster/id
```

The request will return an `HTTP/1.1 200 OK` response and a string that contains the Sensu cluster ID:

```
"23481e76-5844-4d07-b714-6e2ffbbf9315"
```

API Specification

/cluster/id (GET)	
description	Returns the unique Sensu cluster ID.

example url	http://hostname:8080/api/core/v2/cluster/id
query parameters	`timeout`: Defines the timeout when querying etcd. Default is `3`.
response type	String
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
example output	`"23481e76-5844-4d07-b714-6e2ffbbf9315"`

core/v2/clusterrolebindings

***NOTE**: Requests to `core/v2/clusterrolebindings` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all cluster role bindings

The `/clusterrolebindings` API endpoint provides HTTP GET access to cluster role binding data.

Example

The following example demonstrates a GET request to the `/clusterrolebindings` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the cluster role binding definitions:

```
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "cluster-admins"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-admin"
```

```
    },
    "metadata": {
      "name": "cluster-admin",
      "created_by": "admin"
    }
  },
  {
    "subjects": [
      {
        "type": "Group",
        "name": "system:agents"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "system:agent"
    },
    "metadata": {
      "name": "system:agent",
      "created_by": "admin"
    }
  }
]
```

API Specification

/clusterrolebindings (GET)	
description	Returns the list of cluster role bindings.
example url	http://hostname:8080/api/core/v2/clusterrolebindings
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	

- **Success**: 200 (OK)
- **Error**: 500 (Internal Server Error)

---

output


```
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "cluster-admins"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-admin"
    },
    "metadata": {
      "name": "cluster-admin",
      "created_by": "admin"
    }
  },
  {
    "subjects": [
      {
        "type": "Group",
        "name": "system:agents"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "system:agent"
    },
    "metadata": {
      "name": "system:agent"
    }
  }
]
```

Create a new cluster role binding

The `/clusterrolebindings` API endpoint provides HTTP POST access to create a cluster role binding.

Example

In the following example, an HTTP POST request is submitted to the `/clusterrolebindings` API endpoint to create a cluster role binding that assigns the `cluster-admin` cluster role to the user `bob`. The request includes the cluster role binding definition in the request body,

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/clusterrolebindings (POST)

description	Creates a Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings
payload
```
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}
```
 |
| response codes | ¬ **Success**: 201 (Created)
¬ **Malformed**: 400 (Bad Request)
¬ **Error**: 500 (Internal Server Error) |

Get a specific cluster role binding

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP GET access to cluster role binding data for specific `:clusterrolebinding` definitions, by cluster role binding `name`.

Example

The following example queries the `/clusterrolebindings/:clusterrolebinding` API endpoint for the `:clusterrolebinding` named `bob-binder`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/bob-binder \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:clusterrolebinding` definition (in this example, `bob-binder`):

```
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder",
    "created_by": "admin"
  }
}
```

API Specification

/clusterrolebindings/:clusterrolebinding (GET)	
description	Returns the specified cluster role binding.
example url	http://hostname:8080/api/core/v2/clusterrolebindings/bob-binder
response type	Map
response codes	¬ **Success**: 200 (OK) ¬

- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

---

output

```
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder",
    "created_by": "admin"
  }
}
```

Create or update a cluster role binding

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP PUT access to create or update a cluster role binding, by cluster role binding `name`.

Example

In the following example, an HTTP PUT request is submitted to the `/clusterrolebindings/:clusterrolebinding` API endpoint to create a cluster role binding that assigns the `cluster-admin` cluster role to users in the group `ops`. The request includes the cluster role binding definition in the request body:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
```

```
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "ops"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "ops-group-binder"
  }
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-group-binder
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/clusterrolebindings/:clusterrolebinding (PUT)	
description	Creates or updates the specified Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings/ops-group-binder
payload	{ "subjects": [{ "type": "Group", "name": "ops" }], "role_ref": { "type": "ClusterRole", "name": "cluster-admin"

```
  },
  "metadata": {
    "name": "ops-group-binder"
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a cluster role binding with PATCH

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP PATCH access to update `:clusterrolebinding` definitions, specified by cluster role binding name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/clusterrolebindings/:clusterrolebinding` API endpoint to update the subjects array for the `ops-group-binder` check, resulting in a `HTTP/1.1 200 OK` response and the updated cluster role binding definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
```

```
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "ops_team_1"
    },
    {
      "type": "Group",
      "name": "ops_team_2"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-group-binder
```

API Specification

/clusterrolebindings/:clusterrolebinding (PATCH)	
description	Updates the specified Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings/ops-group-binder
payload	{ "subjects": [{ "type": "Group", "name": "ops_team_1" }, { "type": "Group", "name": "ops_team_2" }] }

response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a cluster role binding

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP DELETE access to delete a cluster role binding from Sensu (specified by the cluster role binding name).

Example

The following example shows a request to the `/clusterrolebindings/:clusterrolebinding` API endpoint to delete the check named `ops-binding`, which will result in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-binding
```

API Specification

/clusterrolebindings/:clusterrolebinding (DELETE)	
description	Removes a cluster role binding from Sensu (specified by the cluster role binding name).
example url	http://hostname:8080/api/core/v2/clusterrolebindings/ops-binding
response codes	¬ **Success**: 204 (No Content)

- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get a subset of cluster role bindings with response filtering

The `/clusterrolebindings` API endpoint supports response filtering for a subset of cluster role binding data based on labels and the following fields:

- `clusterrolebinding.name`
- `clusterrolebinding.role_ref.name`
- `clusterrolebinding.role_ref.type`

Example

The following example demonstrates a request to the `/clusterrolebindings` API endpoint with response filtering to retrieve only cluster role binding definitions whose `role_ref.name` includes `cluster-user`:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/clusterrolebindings -G \
--data-urlencode 'fieldSelector="cluster-user" in clusterrolebinding.role_ref.name'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only cluster role binding definitions whose `role_ref.name` includes `cluster-user`:

```
[
  {
    "subjects": [
      {
        "type": "User",
        "name": "ann"
      }
    ],
```

```
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-user"
    },
    "metadata": {
      "name": "ann-binder",
      "created_by": "admin"
    }
  },
  {
    "subjects": [
      {
        "type": "User",
        "name": "bonita"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-user"
    },
    "metadata": {
      "name": "bonita-binder",
      "created_by": "admin"
    }
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/clusterrolebindings (GET) with response filters	
description	Returns the list of cluster role bindings that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/clusterr

	olebindings
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[{ "subjects": [{ "type": "User", "name": "ann" }], "role_ref": { "type": "ClusterRole", "name": "cluster-user" }, "metadata": { "name": "ann-binder", "created_by": "admin" } }, { "subjects": [{ "type": "User", "name": "bonita" }], "role_ref": { "type": "ClusterRole", "name": "cluster-user" }, "metadata": {`

```
      "name": "bonita-binder",
      "created_by": "admin"
    }
  }
]
```

core/v2/clusterroles

***NOTE**: Requests to `core/v2/clusterroles` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all cluster roles

The `/clusterroles` API endpoint provides HTTP GET access to cluster role data.

Example

The following example demonstrates a GET request to the `/clusterroles` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterroles \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the cluster role definitions:

```
[
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "assets",
          "checks",
          "entities",
```

```
          "extensions",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "silenced",
          "roles",
          "rolebindings"
        ],
        "resource_names": null
      },
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "namespaces"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "cluster-admin",
      "created_by": "admin"
```

```
      }
    }
]
```

API Specification

/clusterroles (GET)	
description	Returns the list of cluster roles.
example url	http://hostname:8080/api/core/v2/clusterroles
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "rules": [{ "verbs": ["*"], "resources": ["assets", "checks", "entities", "extensions", "events", "filters", "handlers",

```
            "hooks",
            "mutators",
            "silenced",
            "roles",
            "rolebindings"
          ],
          "resource_names": null
        },
        {
          "verbs": [
            "get",
            "list"
          ],
          "resources": [
            "namespaces"
          ],
          "resource_names": null
        }
      ],
      "metadata": {
        "name": "admin"
      }
    },
    {
      "rules": [
        {
          "verbs": [
            "*"
          ],
          "resources": [
            "*"
          ],
          "resource_names": null
        }
      ],
      "metadata": {
        "name": "cluster-admin",
        "created_by": "admin"
      }
    }
  ]
```

Create a new cluster role

The `/clusterroles` API endpoint provides HTTP POST access to create a cluster role.

Example

In the following example, an HTTP POST request is submitted to the `/clusterroles` API endpoint to create a `global-event-reader` cluster role. The request includes the cluster role definition in the request body:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterroles
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/clusterroles (POST)	
description	Creates a Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles
payload
```
{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```
 |
| response codes | ¬ **Success**: 201 (Created)
¬ **Malformed**: 400 (Bad Request)
¬ **Error**: 500 (Internal Server Error) |

Get a specific cluster role

The `/clusterroles/:clusterrole` API endpoint provides HTTP GET access to cluster role data for specific `:clusterrole` definitions, by cluster role `name`.

Example

The following example queries the `/clusterroles/:clusterrole` API endpoint for the `:clusterrole` named `global-event-reader`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-reader \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:clusterrole` definition (in this example, `global-event-reader`):

```
{
  "metadata": {
    "name": "global-event-reader",
    "created_by": "admin"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

API Specification

/clusterroles/:clusterrole (GET)	
description	Returns the specified cluster role.

example url	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "metadata": { "name": "global-event-reader", "created_by": "admin" }, "rules": [{ "verbs": ["get", "list"], "resources": ["events"], "resource_names": null }] }

Create or update a cluster role

The `/clusterroles/:clusterrole` API endpoint provides HTTP PUT access to create or update a cluster role, by cluster role name.

Example

In the following example, an HTTP PUT request is submitted to the `/clusterroles/:clusterrole` API endpoint to update the `global-event-reader` cluster role by adding `"checks"` to the resources:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "checks",
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterroles
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/clusterroles/:clusterrole (PUT)	
description	Creates or updates the specified Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
payload	

```
{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a cluster role with PATCH

The `/clusterroles/:clusterrole` API endpoint provides HTTP PATCH access to update `:clusterrole` definitions, specified by cluster role name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/clusterroles/:clusterrole` API endpoint to update the verbs array within the rules array for the `global-event-admin` cluster role, resulting in a `HTTP/1.1 200 OK` response and the updated check definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "rules": [
    {
      "verbs": [
        "*"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-admin
```

API Specification

/clusterroles/:clusterrole (PATCH)	
description	Updates the specified Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles/global-event-admin
payload	`{ "rules": [`

```
    {
      "verbs": [
        "*"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a cluster role

The `/clusterroles/:clusterrole` API endpoint provides HTTP DELETE access to delete a cluster role from Sensu (specified by the cluster role name).

Example

The following example shows a request to the `/clusterroles/:clusterrole` API endpoint to delete the cluster role `global-event-reader`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-reader
```

API Specification

/clusterroles/:clusterrole (DELETE)	
description	Removes a cluster role from Sensu (specified by the cluster role name).
example url	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of cluster roles with response filtering

The `/clusterroles` API endpoint supports response filtering for a subset of cluster role data based on labels and the `clusterrole.name` field.

Example

The following example demonstrates a request to the `/clusterroles` API endpoint with response filtering for only cluster role definitions whose `clusterrole.name` includes `admin`:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/clusterroles -G \
--data-urlencode 'fieldSelector=clusterrole.name matches "admin"'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only cluster role definitions whose `clusterrole.name` includes `admin`:

```
[
  {
    "rules": [
      {
```

```
        "verbs": [
          "*"
        ],
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "silenced",
          "roles",
          "rolebindings"
        ],
        "resource_names": null
      },
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "namespaces"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
```

```
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "cluster-admin"
    }
  }
]
```


***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/clusterroles (GET) with response filters	
description	Returns the list of cluster roles that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/clusterroles
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "rules": [ { "verbs": [ "*" ],`

```
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "silenced",
          "roles",
          "rolebindings"
        ],
        "resource_names": null
      },
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "namespaces"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
```

```
    "metadata": {
      "name": "cluster-admin"
    }
  }
]
```

core/v2/entities

***NOTE**: Requests to `core/v2/entities` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all entities

The `/entities` API endpoint provides HTTP GET access to entity data.

Example

The following example demonstrates a GET request to the `/entities` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the entity definitions in the `default` namespace:

```
[
  {
    "entity_class": "agent",
    "sensu_agent_version": "1.0.0",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
```

```
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::f50c:b029:30a5:3e26/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:9f:5d:f3",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe9f:5df3/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
```

```
        },
        {
          "name": "Slack Helper",
          "pid": 1360,
          "ppid": 1349,
          "status": "Ss",
          "background": true,
          "running": true,
          "created": 1582137786,
          "memory_percent": 0.146866455,
          "cpu_percent": 0.308976181461092553
        }
      ]
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    }
  }
]
```

API Specification

/entities (GET)	
description	Returns the list of entities.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "entity_class": "agent", "sensu_agent_version": "1.0.0", "system": { "hostname": "sensu-centos", "os": "linux", "platform": "centos", "platform_family": "rhel", "platform_version": "7.4.1708", "network": { "interfaces": [{ "name": "lo", "addresses": ["127.0.0.1/8", "::1/128"] }, {

```
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
              "10.0.2.15/24",
              "fe80::f50c:b029:30a5:3e26/64"
            ]
          },
          {
            "name": "enp0s8",
            "mac": "08:00:27:9f:5d:f3",
            "addresses": [
              "172.28.128.3/24",
              "fe80::a00:27ff:fe9f:5df3/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "kvm",
      "vm_role": "host",
      "cloud_provider": "",
      "processes": [
        {
          "name": "Slack",
          "pid": 1349,
          "ppid": 0,
          "status": "Ss",
          "background": true,
          "running": true,
          "created": 1582137786,
          "memory_percent": 1.09932518,
          "cpu_percent": 0.3263987595984941
        },
        {
          "name": "Slack Helper",
          "pid": 1360,
          "ppid": 1349,
          "status": "Ss",
          "background": true,
          "running": true,
          "created": 1582137786,
```

```
            "memory_percent": 0.146866455,
            "cpu_percent": 0.308976181461092553
          }
        ]
      },
      "subscriptions": [
        "entity:sensu-centos"
      ],
      "last_seen": 1543349936,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "sensu-centos",
        "namespace": "default",
        "created_by": "admin",
        "labels": null,
        "annotations": null
      }
    }
]
```


Create a new entity

The `/entities` API endpoint provides HTTP POST access to create a Sensu entity.

Example

In the following example, an HTTP POST request is submitted to the `/entities` API endpoint to create a proxy entity named `sensu-centos`. The request includes the entity definition in the request body:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity_class": "proxy",
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "web"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/entities (POST)	
description	Creates a Sensu entity.
example URL	http://hostname:8080/api/core/v2/namespaces/default/entities
payload	<pre>{ "entity_class": "proxy", "sensu_agent_version": "1.0.0",</pre>

```
  "subscriptions": [
    "web"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific entity

The `/entities/:entity` API endpoint provides HTTP GET access to entity data for specific `:entity` definitions, by entity `name`.

Example

The following example queries the `/entities/:entity` API endpoint for the `:entity` named `sensu-centos`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/sensu-centos \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:entity` definition (in this example, `sensu-centos`):

```
{
  "entity_class": "agent",
  "sensu_agent_version": "1.0.0",
  "system": {
    "hostname": "sensu-centos",
    "os": "linux",
    "platform": "centos",
    "platform_family": "rhel",
    "platform_version": "7.4.1708",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::f50c:b029:30a5:3e26/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:9f:5d:f3",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe9f:5df3/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
```

```
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 0.146866455,
        "cpu_percent": 0.308976181461092553
      }
    ]
  },
  "subscriptions": [
    "entity:sensu-centos"
  ],
  "last_seen": 1543349936,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
```

```
  ],
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  }
}
```


API Specification

/entities/:entity (GET)	
description	Returns the specified entity.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "entity_class": "agent", "sensu_agent_version": "1.0.0", "system": { "hostname": "sensu-centos", "os": "linux", "platform": "centos", "platform_family": "rhel", "platform_version": "7.4.1708", "network": { "interfaces": [{

```
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::f50c:b029:30a5:3e26/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:9f:5d:f3",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe9f:5df3/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
```

```
          "name": "Slack Helper",
          "pid": 1360,
          "ppid": 1349,
          "status": "Ss",
          "background": true,
          "running": true,
          "created": 1582137786,
          "memory_percent": 0.146866455,
          "cpu_percent": 0.308976181461092553
        }
      ]
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    }
  }
```

Create or update an entity

> ***NOTE**: This endpoint will not update agent-managed entities.Requests to update agent-managed entities via the Sensu backend REST API will fail and return `HTTP 409 Conflict`.*

The `/entities/:entity` API endpoint provides HTTP PUT access to create or update the specified Sensu entity.

Example

In the following example, an HTTP PUT request is submitted to the `/entities/:entity` API endpoint to update the entity named `sensu-centos`.The request includes the updated entity definition in the request body:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity_class": "proxy",
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "web",
    "system"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/sensu-centos
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the updated entity definition:

```
{
    "entity_class": "proxy",
    "system": {
        "network": {
            "interfaces": null
        },
        "libc_type": "",
        "vm_system": "",
        "vm_role": "",
        "cloud_provider": "",
        "processes": null
    },
    "subscriptions": [
        "web",
        "system"
    ],
    "last_seen": 0,
    "deregister": false,
    "deregistration": {},
    "metadata": {
        "name": "sensu-centos",
        "namespace": "default"
    },
    "sensu_agent_version": "1.0.0"
}
```


API Specification

/entities/:entity (PUT)	
description	Creates or updates the specified Sensu entity. ***NOTE**: When you create an entity via an HTTP PUT request, the entity will use the namespace in the request URL.*
example URL	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos

payload


```
{
  "entity_class": "proxy",
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "web",
    "system"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```


response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update an entity with PATCH

***NOTE**: This endpoint will not update agent-managed entities.Requests to update agent-managed entities via the Sensu backend REST API will fail and return `HTTP 409 Conflict`.*

The `/entities/:entity` API endpoint provides HTTP PATCH access to update **entity configuration attributes** in `:entity` definitions, specified by entity name:

- `labels`
- `annotations`
- `created_by`

- `entity_class`
- `user`
- `subscriptions`
- `deregister`
- `deregistration`
- `redact`
- `keepalive_handler`

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/entities/:entity` API endpoint to add a label for the `sensu-centos` entity, resulting in a `HTTP/1.1 200 OK` response and the updated entity definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/sensu-centos
```

API Specification

/entities/:entity (PATCH)	
description	Updates the specified Sensu entity.
example URL	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
payload
```
{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}
```
 |
| response codes | ¬ **Success**: 200 (OK)
¬ **Malformed**: 400 (Bad Request)
¬ **Error**: 500 (Internal Server Error) |

Delete an entity

The `/entities/:entity` API endpoint provides HTTP DELETE access to delete an entity from Sensu (specified by the entity name).

Example

The following example shows a request to the `/entities/:entity` API endpoint to delete the entity `sensu-centos`, which will result in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/sensu-centos \
```

```
  -H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/entities/:entity (DELETE)	
description	Removes a entity from Sensu (specified by the entity name).
example url	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of entities with response filtering

The `/entities` API endpoint supports response filtering for a subset of entity data based on labels and the following fields:

- `entity.name`
- `entity.namespace`
- `entity.deregister`
- `entity.entity_class`
- `entity.subscriptions`

Example

The following example demonstrates a request to the `/entities` API endpoint with response filtering for only entity definitions whose subscriptions include `linux`:

```
```

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector="linux" in entity.subscriptions'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only entity definitions whose subscriptions include `linux`:


```
[
  {
    "entity_class": "agent",
    "system": {
      "network": {
        "interfaces": null
      },
      "libc_type": "",
      "vm_system": "",
      "vm_role": "",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "linux",
      "entity:datastore01"
    ],
    "last_seen": 0,
    "deregister": false,
    "deregistration": {},
    "metadata": {
      "name": "datastore01",
      "namespace": "default",
      "labels": {
        "region": "us-west-1",
        "service_type": "datastore",
        "sensu.io/managed_by": "sensuctl"
      }
    },
    "sensu_agent_version": ""
  },
  {
    "entity_class": "agent",
```

```
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::c68e:8fd8:32f0:7c5d/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:3b:a9:9f",
            "addresses": [
              "192.168.56.23/24",
              "fe80::a00:27ff:fe3b:a99f/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "linux",
```

```
      "entity:sensu-centos"
    ],
    "last_seen": 1644615964,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default"
    },
    "sensu_agent_version": "6.6.5"
  }
]
```


***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/entities (GET) with response filters	
description	Returns the list of entities that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/entities
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.

response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output
```
[
  {
    "entity_class": "agent",
    "system": {
      "network": {
        "interfaces": null
      },
      "libc_type": "",
      "vm_system": "",
      "vm_role": "",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "linux",
      "entity:datastore01"
    ],
    "last_seen": 0,
    "deregister": false,
    "deregistration": {},
    "metadata": {
      "name": "datastore01",
      "namespace": "default",
      "labels": {
        "region": "us-west-1",
        "service_type": "datastore",
        "sensu.io/managed_by": "sensuctl"
      }
    },
    "sensu_agent_version": ""
  },
  {
    "entity_class": "agent",
```
 |

```
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::c68e:8fd8:32f0:7c5d/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:3b:a9:9f",
            "addresses": [
              "192.168.56.23/24",
              "fe80::a00:27ff:fe3b:a99f/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "linux",
```

```
      "entity:sensu-centos"
    ],
    "last_seen": 1644615964,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default"
    },
    "sensu_agent_version": "6.6.5"
  }
]
```

core/v2/events

***PRO TIP**: The `core/v2/events` API endpoints are primarily designed to provide HTTP access to event data created by agent-executed checks. To test your Sensu observability pipeline, use the agent API to create new ad hoc events or sensuctl or the web UI to execute existing checks on demand.*

***NOTE**: Requests to `core/v2/events` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all events

The `/events` API endpoint provides HTTP GET access to event data.

Example

The following example demonstrates a request to the `/events` API endpoint, resulting in a JSON array that contains event definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the event definitions in the `default` namespace:

```
[
  {
    "check": {
      "command": "check-cpu-usage -w 75 -c 90",
```

```
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 60,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2 nice=0.0 system=0.0 idle=99.8
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
```

```
    "metadata": {
      "name": "check_cpu",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "processed_by": "server1",
    "scheduler": "memory"
  },
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
```

```
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313661,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "server1",
      "namespace": "default"
    },
    "sensu_agent_version": "6.2.7"
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
```

```
    "metadata": {
      "namespace": "default"
    },
    "sequence": 3,
    "timestamp": 1620313666
  }
]
```

API Specification

/events (GET)	
description	Returns the list of events.
example url	http://hostname:8080/api/core/v2/namespaces/default/events
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "check": { "command": "check-cpu-usage -w 75 -c 90", "handlers": [], "high_flap_threshold": 0, "interval": 60, "low_flap_threshold": 0, "publish": true, "runtime_assets": ["check-cpu-usage"

```
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2
nice=0.0 system=0.0 idle=99.8 iowait=0.0 irq=0.0
softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
```

```
      "processed_by": "server1",
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::bc00:e2c8:1059:3868/64"
              ]
            },
            {
              "name": "eth1",
              "mac": "08:00:27:73:87:93",
              "addresses": [
                "172.28.128.57/24",
                "fe80::a00:27ff:fe73:8793/64"
              ]
            }
          ]
        },
        "arch": "amd64",
        "libc_type": "glibc",
        "vm_system": "vbox",
        "vm_role": "guest",
```

```
        "cloud_provider": "",
        "processes": null
      },
      "subscriptions": [
        "system",
        "entity:server1"
      ],
      "last_seen": 1620313661,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "server1",
        "namespace": "default"
      },
      "sensu_agent_version": "6.2.7"
    },
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ],
    "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
    "metadata": {
      "namespace": "default"
    },
    "sequence": 3,
    "timestamp": 1620313666
  }
```

```
]
```

Create a new event

The `/events` API endpoint provides HTTP POST access to create an event and send it to the Sensu observability pipeline.

Example

In the following example, an HTTP POST request is submitted to the `/events` API endpoint to create an event. The request includes information about the check and entity represented by the event:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "state": "failing",
    "status": 2,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
```

```
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events
```

The request will return a successful `HTTP/1.1 201 Created` response.

To create useful, actionable events, we recommend using check attributes like `status` (`0` for OK, `1` for warning, `2` for critical) and `output`, as well as adding `pipelines`, as shown in this example For more information about event attributes and their available values, read the event specification.

For events created with this endpoint, the following attributes have the default value `0` unless you specify a different value for testing:

- `executed`
- `issued`
- `last_seen`
- `status`

The `last_ok` attribute will default to `0` even if you manually specify OK status in the request body.

The `sensu_agent_version` attribute will return with a null value for events created with this endpoint because these events are not created by an agent-executed check.

API Specification

/events (POST)	
description	Creates a new Sensu event. To update an existing event, use the `/events` PUT endpoint. If you create a new event that references an entity that does not already exist, sensu-backend will automatically create a proxy entity in the same namespace when the event is published. If you create an event that references an existing entity but includes different information for entity attributes, Sensu **will not** make any changes to the existing entity's definition based on the event you create via the API.

***NOTE**: An agent cannot belong to, execute checks in, or create events in more than one namespace.*

example URL	http://hostname:8080/api/core/v2/namespaces/default/events

payload


```
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "state": "failing",
    "status": 2,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ]
}
```


response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get event data for a specific entity

The `/events/:entity` API endpoint provides HTTP GET access to event data specific to an `:entity`, by entity `name`.

Example

The following example queries the `/events/:entity` API endpoint for Sensu events for the `server1` entity:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1 \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the Sensu events for the `server1` entity:


```
[
  {
    "check": {
      "command": "check-cpu-usage -w 75 -c 90",
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 60,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
```

```
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2 nice=0.0 system=0.0 idle=99.8
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "processed_by": "server1",
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
```

```
    "platform_family": "rhel",
    "platform_version": "7.5.1804",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "eth0",
          "mac": "08:00:27:8b:c9:3f",
          "addresses": [
            "10.0.2.15/24",
            "fe80::bc00:e2c8:1059:3868/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:73:87:93",
          "addresses": [
            "172.28.128.57/24",
            "fe80::a00:27ff:fe73:8793/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "system",
    "entity:server1"
  ],
  "last_seen": 1620313661,
  "deregister": false,
```

```
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "server1",
        "namespace": "default"
      },
      "sensu_agent_version": "6.2.7"
    },
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ],
    "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
    "metadata": {
      "namespace": "default"
    },
    "sequence": 3,
    "timestamp": 1620313666
  },
  {
    "check": {
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 20,
      "low_flap_threshold": 0,
      "publish": false,
      "runtime_assets": null,
      "subscriptions": [],
```

```
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 120,
    "round_robin": false,
    "executed": 1620313714,
    "history": [
      {
        "status": 0,
        "executed": 1620313314
      },
      {
        "status": 0,
        "executed": 1620313334
      },
      {
        "status": 0,
        "executed": 1620313354
      },
      {
        "...": 0,
        "...": 1620313374
      }
    ],
    "issued": 1620313714,
    "output": "Keepalive last sent from server1 at 2021-05-06 15:08:34 +0000 UTC",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1620313714,
    "occurrences": 358,
    "occurrences_watermark": 358,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "keepalive",
      "namespace": "default"
    },
    "secrets": null,
```

```
      "is_silenced": false,
      "processed_by": "server1",
      "scheduler": "etcd"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::bc00:e2c8:1059:3868/64"
              ]
            },
            {
              "name": "eth1",
              "mac": "08:00:27:73:87:93",
              "addresses": [
                "172.28.128.57/24",
                "fe80::a00:27ff:fe73:8793/64"
              ]
            }
          ]
        },
        "arch": "amd64",
        "libc_type": "glibc",
        "vm_system": "vbox",
```

```
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313714,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "server1",
      "namespace": "default"
    },
    "sensu_agent_version": "6.2.7"
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "id": "8717b1dc-47d2-4b73-a259-ee2645cadbf5",
  "metadata": {
    "namespace": "default"
  },
  "sequence": 359,
  "timestamp": 1620313714
```

```
    }
]
```

API Specification

/events/:entity (GET)	
description	Returns a list of events for the specified entity.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	[{ "check": { "command": "check-cpu-usage -w 75 -c 90", "handlers": [], "high_flap_threshold": 0, "interval": 60, "low_flap_threshold": 0, "publish": true, "runtime_assets": ["check-cpu-usage"], "subscriptions": ["system"], "proxy_entity_name": "", "check_hooks": null,

```
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2
nice=0.0 system=0.0 idle=99.8 iowait=0.0 irq=0.0
softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "processed_by": "server1",
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
```

```
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
```

```
      ],
      "last_seen": 1620313661,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "server1",
        "namespace": "default"
      },
      "sensu_agent_version": "6.2.7"
    },
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ],
    "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
    "metadata": {
      "namespace": "default"
    },
    "sequence": 3,
    "timestamp": 1620313666
  },
  {
    "check": {
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 20,
      "low_flap_threshold": 0,
```

```
      "publish": false,
      "runtime_assets": null,
      "subscriptions": [],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 120,
      "round_robin": false,
      "executed": 1620313714,
      "history": [
        {
          "status": 0,
          "executed": 1620313314
        },
        {
          "status": 0,
          "executed": 1620313334
        },
        {
          "status": 0,
          "executed": 1620313354
        },
        {
          "...": 0,
          "...": 1620313374
        }
      ],
      "issued": 1620313714,
      "output": "Keepalive last sent from server1 at
2021-05-06 15:08:34 +0000 UTC",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313714,
      "occurrences": 358,
      "occurrences_watermark": 358,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
```

```
      "name": "keepalive",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "processed_by": "server1",
    "scheduler": "etcd"
  },
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
        ]
```

```
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313714,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "server1",
      "namespace": "default"
    },
    "sensu_agent_version": "6.2.7"
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "id": "8717b1dc-47d2-4b73-a259-ee2645cadbf5",
  "metadata": {
```

```
      "namespace": "default"
    },
    "sequence": 359,
    "timestamp": 1620313714
  }
]
```


Get event data for a specific entity and check

The `/events/:entity/:check` API endpoint provides HTTP GET access to event data for the specified entity and check.

Example

In the following example, an HTTP GET request is submitted to the `/events/:entity/:check` API endpoint to retrieve the event for the `server1` entity and the `check_cpu` check:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/check_cpu \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the Sensu events for the `server1` entity and `check_cpu` check:


```
{
  "check": {
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
```

```
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 5.050929017,
    "executed": 1620313539,
    "history": null,
    "issued": 1620313539,
    "output": "CheckCPU TOTAL OK: total=2.85 user=2.65 nice=0.0 system=0.2
idle=97.15 iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1620313539,
    "occurrences": 1,
    "occurrences_watermark": 1,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "processed_by": "server1",
    "scheduler": ""
  },
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
```

```
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "eth0",
          "mac": "08:00:27:8b:c9:3f",
          "addresses": [
            "10.0.2.15/24",
            "fe80::bc00:e2c8:1059:3868/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:73:87:93",
          "addresses": [
            "172.28.128.57/24",
            "fe80::a00:27ff:fe73:8793/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "system",
    "entity:server1"
  ],
  "last_seen": 1620313539,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
```

```
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "server1",
      "namespace": "default"
    },
    "sensu_agent_version": "6.2.7"
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ],
  "id": "9a9c7515-0a04-43f3-9351-d8da88942b1b",
  "metadata": {
    "namespace": "default"
  },
  "sequence": 1,
  "timestamp": 1620313546
}
```


API Specification

/events/:entity/:check (GET)	
description	Returns an event for the specified entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1/check_cpu

response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "check": { "command": "check-cpu-usage -w 75 -c 90", "handlers": [], "high_flap_threshold": 0, "interval": 60, "low_flap_threshold": 0, "publish": true, "runtime_assets": ["check-cpu-usage"], "subscriptions": ["system"], "proxy_entity_name": "", "check_hooks": null, "stdin": false, "subdue": null, "ttl": 0, "timeout": 0, "round_robin": false, "duration": 5.050929017, "executed": 1620313539, "history": null, "issued": 1620313539, "output": "CheckCPU TOTAL OK: total=2.85 user=2.65 nice=0.0 system=0.2 idle=97.15 iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n", "state": "passing", "status": 0, "total_state_change": 0,

```
    "last_ok": 1620313539,
    "occurrences": 1,
    "occurrences_watermark": 1,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check_cpu",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "processed_by": "server1",
    "scheduler": ""
  },
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
```

```
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313539,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "server1",
      "namespace": "default"
    },
    "sensu_agent_version": "6.2.7"
  },
  "pipelines": [
    {
```

```
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "incident_alerts"
      }
    ],
    "id": "9a9c7515-0a04-43f3-9351-d8da88942b1b",
    "metadata": {
      "namespace": "default"
    },
    "sequence": 1,
    "timestamp": 1620313546
  }
```


Create a new event for an entity and check

The `/events/:entity/:check` API endpoint provides HTTP POST access to create an event and send it to the Sensu observability pipeline.

Example

In the following example, an HTTP POST request is submitted to the `/events/:entity/:check` API endpoint to create an event for the `server1` entity and the `server-health` check and process it using the `incident_alerts` pipeline. The event includes a status code of `1`, indicating a warning, and an output message of `Server error`.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
```

```
    "output": "Server error",
    "status": 1,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ]
} \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The request will return a successful `HTTP/1.1 201 Created` response.

> ***NOTE**: A namespace is not required to create the event. The event will use the namespace in the URL by default.*

You can use sensuctl or the Sensu web UI to view the event:

```
sensuctl event list
```

The response should list the event with the status and output specified in the request:

```
   Entity      Check           Output          Status  Silenced       Timestamp
──────────── ──────────── ──────────────── ──────── ────────── ──────────────────────────────
   server1     server-health   Server error          1     false      2019-03-14 16:56:09 +0000 UTC
```

For events created with this endpoint, the following attributes have the default value `0` unless you specify a different value for testing:

- `executed`
- `issued`
- `last_seen`
- `status`

The `last_ok` attribute will default to `0` even if you manually specify OK status in the request body.

The `sensu_agent_version` attribute will return with a null value for events created with this endpoint because these events are not created by an agent-executed check.

API Specification

/events/:entity/:check (POST)	
description	Creates an event for the specified entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1/server-health
payload	

```
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
```

```
      "name": "incident_alerts"
    }
  ]
}
```

---

response codes

- **Success**: 201 (Created)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Create or update an event for an entity and check

The `/events/:entity/:check` API endpoint provides HTTP PUT access to create or update an event and send it to the Sensu observability pipeline.

Example

In the following example, an HTTP PUT request is submitted to the `/events/:entity/:check` API endpoint to create an event for the `server1` entity and the `server-health` check and process it using the `incident_alerts` pipeline. The event includes a status code of `1`, indicating a warning, and an output message of `Server error`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
```

```
    "status": 1,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The request will return a successful `HTTP/1.1 201 Created` response.

> ***NOTE**: A namespace is not required to create the event. The event will use the namespace in the URL by default.*

You can use sensuctl or the Sensu web UI to view the event:

```
sensuctl event list
```

The response should list the event with the status and output specified in the request:

```
   Entity      Check           Output          Status  Silenced       Timestamp
──────────── ──────────── ──────────────────── ──────── ────────── ──────────────────────────────
   server1   server-health  Server error           1     false     2019-03-14 16:56:09 +0000 UTC
```

API Specification

/events/:entity/:check (PUT)	
description	Creates an event for the specified entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1/server-health
payload
```
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ]
}
```
 |
| payload parameters | Review the payload parameters section below. |
| response codes | ¬ **Success**: 201 (Created)
¬ **Missing**: 404 (Not Found)
¬ **Error**: 500 (Internal Server Error) |

Payload parameters

The `/events/:entity/:check` PUT endpoint requires a request payload that contains an `entity` scope and a `check` scope.

- The `entity` scope contains information about the component of your infrastructure represented by the event. At minimum, Sensu requires the `entity` scope to contain the `entity_class` (`agent` or `proxy`) and the entity `name` and `namespace` within a `metadata` scope. For more information about entity attributes, review the entity specification.
- The `check` scope contains information about the event status and how the event was created. At minimum, Sensu requires the `check` scope to contain a `name` within a `metadata` scope and either an `interval` or `cron` attribute. For more information about check attributes, review the check specification.

Example request with minimum required event attributes

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1"
    }
  },
  "check": {
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The request will return a successful `HTTP/1.1 201 Created` response.

The minimum required attributes let you create an event using the `/events/:entity/:check` PUT endpoint, but the request can include any attributes defined in the event specification. To create useful,

actionable events, we recommend adding check attributes such as the event `status` (`0` for OK, `1` for warning, `2` for critical), an `output` message, and one or more `pipelines` .For more information about these attributes and their available values, review the event specification.

Example request with minimum recommended event attributes

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "incident_alerts"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The request will return a successful `HTTP/1.1 201 Created` response.

Create metrics events

In addition to the `entity` and `check` scopes, Sensu events can include a `metrics` scope that contains metrics in Sensu metric format.Read the events reference and for more information about Sensu metric format.

Example request including metrics

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "status": 0,
    "output_metric_handlers": ["influxdb"],
    "interval": 60,
    "metadata": {
      "name": "server-metrics"
    }
  },
  "metrics": {
    "handlers": [
      "influxdb"
    ],
    "points": [
      {
        "name": "server1.server-metrics.time_total",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "server1.server-metrics.time_namelookup",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.004
```

```
      }
    ]
  },
  "pipelines": [
    {
      "api_version": "core/v2",
      "type": "Pipeline",
      "name": "metrics_workflows"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-metrics
```

The request will return a successful `HTTP/1.1 201 Created` response.

Delete an event

Example

The following example shows a request to the `/events/:entity/:check` API endpoint to delete the event produced by the `server1` entity and `check_cpu` check, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/check_cpu \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/events/:entity/:check (DELETE)	
description	Deletes the event created by the specified entity using the specified check.
example url	http://hostname:8080/api/core/v2/namespaces/default/eve

response codes

- **Success**: 204 (No Content)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get a subset of events with response filtering

The `/events` API endpoint supports response filtering for a subset of event data based on labels and the following fields:

- `event.name`
- `event.namespace`
- `event.is_silenced`
- `event.check.handlers`
- `event.check.is_silenced`
- `event.check.name`
- `event.check.publish`
- `event.check.round_robin`
- `event.check.runtime_assets`
- `event.check.status`
- `event.check.subscriptions`
- `event.entity.deregister`
- `event.entity.entity_class`
- `event.entity.name`
- `event.entity.subscriptions`

Example

The following example demonstrates a request to the `/events` API endpoint with response filtering

for events from entities whose subscriptions include `linux`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector="linux" in event.entity.subscriptions'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only event definitions for entities whose subscriptions include `linux`:


```
[
  {
    "timestamp": 1644848031,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "sensu-centos",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::c68e:8fd8:32f0:7c5d/64"
              ]
            },
            {
              "name": "eth1",
```

```
            "mac": "08:00:27:3b:a9:9f",
            "addresses": [
              "192.168.56.23/24",
              "fe80::a00:27ff:fe3b:a99f/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "linux",
      "entity:sensu-centos",
      "system"
    ],
    "last_seen": 1644848029,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin"
    },
    "sensu_agent_version": "6.6.5"
  },
```

```
    "check": {
      "command": "check-cpu-usage -w 1 -c 2",
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 15,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 2.010462294,
      "executed": 1644848029,
      "history": [
        {
          "status": 2,
          "executed": 1644847740
        },
        {
          "status": 1,
          "executed": 1644847755
        },
        {
          "status": 1,
          "executed": 1644847770
        },
        {
          "status": 2,
          "executed": 1644847785
        },
        {
          "status": 2,
          "executed": 1644847800
```

```
        },
        {
          "status": 1,
          "executed": 1644847815
        },
        {
          "status": 2,
          "executed": 1644847830
        },
        {
          "status": 1,
          "executed": 1644847845
        },
        {
          "status": 1,
          "executed": 1644847860
        },
        {
          "status": 1,
          "executed": 1644847875
        },
        {
          "status": 1,
          "executed": 1644847890
        },
        {
          "status": 0,
          "executed": 1644847905
        },
        {
          "status": 2,
          "executed": 1644847920
        },
        {
          "status": 1,
          "executed": 1644847935
        },
        {
          "status": 0,
          "executed": 1644847950
        },
        {
```

```
          "status": 0,
          "executed": 1644847965
        },
        {
          "status": 2,
          "executed": 1644847980
        },
        {
          "status": 2,
          "executed": 1644847995
        },
        {
          "status": 1,
          "executed": 1644848010
        },
        {
          "status": 1,
          "executed": 1644848014
        },
        {
          "status": 0,
          "executed": 1644848029
        }
      ],
      "issued": 1644848029,
      "output": "check-cpu-usage OK: 0.51% CPU usage | cpu_idle=99.49,
cpu_system=0.51, cpu_user=0.00, cpu_nice=0.00, cpu_iowait=0.00, cpu_irq=0.00,
cpu_softirq=0.00, cpu_steal=0.00, cpu_guest=0.00, cpu_guestnice=0.00\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 59,
      "last_ok": 1644848029,
      "occurrences": 1,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "secrets": null,
```

```
      "is_silenced": false,
      "scheduler": "memory",
      "processed_by": "sensu-centos",
      "pipelines": []
    },
    "metadata": {
      "namespace": "default"
    },
    "id": "f5ef6190-a8e2-4660-9ad1-02ae0a2e89f4",
    "sequence": 2,
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "metrics_workflows"
      }
    ]
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/events (GET) with response filters	
description	Returns the list of events that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/events
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK)

↰ **Error**: 500 (Internal Server Error)

---

output


```
[
  {
    "timestamp": 1644848031,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "sensu-centos",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",

"fe80::c68e:8fd8:32f0:7c5d/64"
              ]
            },
            {
              "name": "eth1",
              "mac": "08:00:27:3b:a9:9f",
              "addresses": [
                "192.168.56.23/24",

"fe80::a00:27ff:fe3b:a99f/64"
              ]
            }
```

```
          ]
        },
        "arch": "amd64",
        "libc_type": "glibc",
        "vm_system": "vbox",
        "vm_role": "guest",
        "cloud_provider": "",
        "processes": null
      },
      "subscriptions": [
        "linux",
        "entity:sensu-centos",
        "system"
      ],
      "last_seen": 1644848029,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "sensu-centos",
        "namespace": "default",
        "created_by": "admin"
      },
      "sensu_agent_version": "6.6.5"
    },
    "check": {
      "command": "check-cpu-usage -w 1 -c
2",
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 15,
```

```
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 2.010462294,
      "executed": 1644848029,
      "history": [
        {
          "status": 2,
          "executed": 1644847740
        },
        {
          "status": 1,
          "executed": 1644847755
        },
        {
          "status": 1,
          "executed": 1644847770
        },
        {
          "status": 2,
          "executed": 1644847785
        },
        {
          "status": 2,
          "executed": 1644847800
        },
        {
          "status": 1,
          "executed": 1644847815
        },
```

```
        {
          "status": 2,
          "executed": 1644847830
        },
        {
          "status": 1,
          "executed": 1644847845
        },
        {
          "status": 1,
          "executed": 1644847860
        },
        {
          "status": 1,
          "executed": 1644847875
        },
        {
          "status": 1,
          "executed": 1644847890
        },
        {
          "status": 0,
          "executed": 1644847905
        },
        {
          "status": 2,
          "executed": 1644847920
        },
        {
          "status": 1,
          "executed": 1644847935
        },
        {
          "status": 0,
          "executed": 1644847950
        },
        {
          "status": 0,
          "executed": 1644847965
        },
        {
          "status": 2,
```

```
          "executed": 1644847980
        },
        {
          "status": 2,
          "executed": 1644847995
        },
        {
          "status": 1,
          "executed": 1644848010
        },
        {
          "status": 1,
          "executed": 1644848014
        },
        {
          "status": 0,
          "executed": 1644848029
        }
      ],
      "issued": 1644848029,
      "output": "check-cpu-usage OK: 0.51%
CPU usage | cpu_idle=99.49, cpu_system=0.51,
cpu_user=0.00, cpu_nice=0.00,
cpu_iowait=0.00, cpu_irq=0.00,
cpu_softirq=0.00, cpu_steal=0.00,
cpu_guest=0.00, cpu_guestnice=0.00\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 59,
      "last_ok": 1644848029,
      "occurrences": 1,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "scheduler": "memory",
```

```
      "processed_by": "sensu-centos",
      "pipelines": []
    },
    "metadata": {
      "namespace": "default"
    },
    "id": "f5ef6190-a8e2-4660-9ad1-
02ae0a2e89f4",
    "sequence": 2,
    "pipelines": [
      {
        "api_version": "core/v2",
        "type": "Pipeline",
        "name": "metrics_workflows"
      }
    ]
  }
]
```

core/v2/filters

***NOTE**: Requests to `core/v2/filters` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all event filters

The `/filters` API endpoint provides HTTP GET access to event filter data.

Example

The following example demonstrates a GET request to the `/filters` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters \
-H "Authorization: Bearer $TOKEN"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the event filter definitions in the `default` namespace:

```
[
  {
    "metadata": {
      "name": "development_filter",
       "namespace": "default",
       "created_by": "admin"
    },
    "action": "deny",
    "expressions": [
      "event.entity.metadata.namespace == 'development'"
    ],
```

```
    "runtime_assets": null
  },
  {
    "metadata": {
      "name": "state_change_only",
      "namespace": "default"
    },
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": null
  }
]
```

API Specification

/filters (GET)	
description	Returns the list of event filters.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "metadata": { "name": "development_filter",`

```
        "namespace": "default",
        "created_by": "admin"
      },
      "action": "deny",
      "expressions": [
        "event.entity.metadata.namespace == 'development'"
      ],
      "runtime_assets": null
    },
    {
      "metadata": {
        "name": "state_change_only",
        "namespace": "default"
      },
      "action": "allow",
      "expressions": [
        "event.check.occurrences == 1"
      ],
      "runtime_assets": null
    }
  ]
```

Create a new event filter

The `/filters` API endpoint provides HTTP POST access to create an event filter.

Example

In the following example, an HTTP POST request is submitted to the `/filters` API endpoint to create the event filter `development_filter`.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "development_filter",
```

```
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "action": "deny",
  "expressions": [
    "event.entity.metadata.namespace == 'development'"
  ],
  "runtime_assets": []
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/filters (POST)	
description	Creates a Sensu event filter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/filters
payload	<pre>{ "metadata": { "name": "development_filter", "namespace": "default", "labels": null, "annotations": null }, "action": "deny", "expressions": ["event.entity.metadata.namespace == 'development'"], "runtime_assets": [] }</pre>
response codes	¬ **Success**: 201 (Created)

- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific event filter

The `/filters/:filter` API endpoint provides HTTP GET access to event filter data for specific `:filter` definitions, by filter name.

Example

The following example queries the `/filters/:filter` API endpoint for the `:filter` named `state_change_only`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/state_change_only \
-H "Authorization: Bearer $TOKEN"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:filter` definition (in this example, `state_change_only`):

```
{
  "metadata": {
    "name": "state_change_only",
    "namespace": "default",
    "created_by": "admin"
  },
  "action": "allow",
  "expressions": [
    "event.check.occurrences == 1"
  ],
  "runtime_assets": null
}
```

API Specification

/filters/:filter (GET)	
description	Returns the specified event filter.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters/state_change_only
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	`{ "metadata": { "name": "state_change_only", "namespace": "default", "created_by": "admin" }, "action": "allow", "expressions": [ "event.check.occurrences == 1" ], "runtime_assets": null }`

Create or update an event filter

The `/filters/:filter` API endpoint provides HTTP PUT access to create or update an event filter.

Example

In the following example, an HTTP PUT request is submitted to the `/filters` API endpoint to create the event filter `development_filter`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "development_filter",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "action": "deny",
  "expressions": [
    "event.entity.metadata.namespace == 'development'"
  ],
  "runtime_assets": []
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/development_filter
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/filters/:filter (PUT)	
description	Creates or updates the specified Sensu event filter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/filters/development_filter
payload	`{ "metadata": { "name": "development_filter", "namespace": "default", "labels": null, "annotations": null`

```
  },
  "action": "deny",
  "expressions": [
    "event.entity.metadata.namespace == 'development'"
  ],
  "runtime_assets": []
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a filter with PATCH

The `/filters/:filter` API endpoint provides HTTP PATCH access to update `:filter` definitions, specified by filter name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/filters/:filter` API endpoint to update the expressions array for the `us-west` filter, resulting in a `HTTP/1.1 200 OK` response and the updated event filter definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
```

```
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "expressions": [
    "event.check.occurrences == 3"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filter/us-west
```

API Specification

/filters/:filter (PATCH)	
description	Updates the specified Sensu filter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/filter/us-west
payload	<pre>{ "expressions": ["event.check.occurrences == 3"] }</pre>
response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete an event filter

The `/filters/:filter` API endpoint provides HTTP DELETE access to delete an event filter from Sensu (specified by the filter name).

Example

The following example shows a request to the `/filters/:filter` API endpoint to delete the event filter `development_filter`, which will result in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/development_filter \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/filters/:filter (DELETE)	
description	Removes the specified event filter from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters/development_filter
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of filters with response filtering

The `/filters` API endpoint supports response filtering for a subset of filter data based on labels and the following fields:

- `filter.name`
- `filter.namespace`
- `filter.action`
- `filter.runtime_assets`

Example

The following example demonstrates a request to the `/filters` API endpoint with response filtering for only filter definitions whose action is `allow`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/filters
-G \
--data-urlencode 'fieldSelector=filter.action == allow'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only event filter definitions whose action is `allow`:


```
[
  {
    "metadata": {
      "name": "filter_interval_60_hourly",
      "namespace": "default",
      "created_by": "admin"
    },
    "action": "allow",
    "expressions": [
      "event.check.interval == 60",
      "event.check.occurrences == 1 || event.check.occurrences % 60 == 0"
    ],
    "runtime_assets": null
  },
  {
    "metadata": {
      "name": "state_change_only",
      "namespace": "default",
      "created_by": "admin"
    },
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": null
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/filters (GET) with response filters	
description	Returns the list of filters that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/filters
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output
```
[
  {
    "metadata": {
      "name": "filter_interval_60_hourly",
      "namespace": "default",
      "created_by": "admin"
    },
    "action": "allow",
    "expressions": [
      "event.check.interval == 60",
      "event.check.occurrences == 1 ||
event.check.occurrences % 60 == 0"
    ],
    "runtime_assets": null
  },
```
 |

```
  {
    "metadata": {
      "name": "state_change_only",
      "namespace": "default",
      "created_by": "admin"
    },
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": null
  }
]
```

core/v2/handlers

***NOTE**: Requests to `core/v2/handlers` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all handlers

The `/handlers` API endpoint provides HTTP GET access to handler data.

Example

The following example demonstrates a GET request to the `/handlers` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the handler definitions in the `default` namespace:


```
[
  {
    "metadata": {
      "name": "influx-db",
      "namespace": "default",
      "created_by": "admin"
    },
    "type": "pipe",
    "command": "sensu-influxdb-handler -d sensu",
    "timeout": 0,
    "handlers": null,
```

```
    "filters": null,
    "env_vars": [
      "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
      "INFLUXDB_USER=sensu",
      "INFLUXDB_PASSWORD=password"
    ],
    "runtime_assets": ["sensu/sensu-influxdb-handler"]
  },
  {
    "metadata": {
      "name": "slack",
      "namespace": "default",
      "created_by": "admin"
    },
    "type": "pipe",
    "command": "sensu-slack-handler --channel '#monitoring'",
    "timeout": 0,
    "handlers": null,
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXXX"
    ],
    "runtime_assets": ["sensu/sensu-influxdb-handler"]
  }
]
```

API Specification

/handlers (GET)	
description	Returns the list of handlers.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers
pagination	This endpoint supports pagination using the `limit` and `continue`

	query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "metadata": { "name": "influx-db", "namespace": "default", "created_by": "admin" }, "type": "pipe", "command": "sensu-influxdb-handler -d sensu", "timeout": 0, "handlers": null, "filters": null, "env_vars": ["INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086", "INFLUXDB_USER=sensu", "INFLUXDB_PASSWORD=password"], "runtime_assets": ["sensu/sensu-influxdb-handler"] }, { "metadata": { "name": "slack", "namespace": "default" }, "type": "pipe", "command": "sensu-slack-handler --channel '#monitoring'", "timeout": 0,

```
    "handlers": null,
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T000000
00/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
    ],
    "runtime_assets": ["sensu/sensu-slack-handler"]
  }
]
```

Create a new handler

The `/handlers` API endpoint provides HTTP POST access to create a handler.

Example

In the following example, an HTTP POST request is submitted to the `/handlers` API endpoint to create the event handler `influx-db`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "influx-db",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-influxdb-handler -d sensu",
  "env_vars": [
    "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
    "INFLUXDB_USER=sensu",
```

```
    "INFLUXDB_PASSWORD=password"
  ],
  "filters": [],
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/handlers (POST)	
description	Creates a Sensu handler.
example URL	http://hostname:8080/api/core/v2/namespaces/default/handlers
payload	{ "metadata": { "name": "influx-db", "namespace": "default", "labels": null, "annotations": null }, "command": "sensu-influxdb-handler -d sensu", "env_vars": ["INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086", "INFLUXDB_USER=sensu", "INFLUXDB_PASSWORD=password"], "filters": [], "handlers": [], "runtime_assets": [], "timeout": 0,

```
  "type": "pipe"
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific handler

The `/handlers/:handler` API endpoint provides HTTP GET access to handler data for specific `:handler` definitions, by handler name.

Example

The following example queries the `/handlers/:handler` API endpoint for the `:handler` named `slack`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/slack \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:handler` definition (in this example, `slack`):

```
{
  "metadata": {
    "name": "slack",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
```

```
  "command": "sensu-slack-handler --channel '#monitoring'",
  "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
  ],
  "filters": [
    "is_incident",
    "not_silenced"
  ],
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
}
```

API Specification

/handlers/:handler (GET)	
description	Returns a handler.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers/slack
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "metadata": { "name": "slack", "namespace": "default", "created_by": "admin", "labels": null,

```
    "annotations": null
  },
  "command": "sensu-slack-handler --channel
'#monitoring'",
  "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T
00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
  ],
  "filters": [
    "is_incident",
    "not_silenced"
  ],
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
}
```

Create or update a handler

The `/handlers/:handler` API endpoint provides HTTP PUT access to create or update a specific `:handler` definition, by handler name.

Example

In the following example, an HTTP PUT request is submitted to the `/handlers/:handler` API endpoint to create the handler `influx-db`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "influx-db",
    "namespace": "default",
    "labels": null,
```

```
    "annotations": null
  },
  "command": "sensu-influxdb-handler -d sensu",
  "env_vars": [
    "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
    "INFLUXDB_USER=sensu",
    "INFLUXDB_PASSWORD=password"
  ],
  "filters": [],
  "handlers": [],
  "runtime_assets": ["sensu/sensu-influxdb-handler"],
  "timeout": 0,
  "type": "pipe"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/influx-db
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/handlers/:handler (PUT)	
description	Creates or updates the specified Sensu handler.
example URL	http://hostname:8080/api/core/v2/namespaces/default/handlers/influx-db
payload	{ "metadata": { "name": "influx-db", "namespace": "default", "labels": null, "annotations": null }, "command": "sensu-influxdb-handler -d sensu", "env_vars": ["INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",

```
      "INFLUXDB_USER=sensu",
      "INFLUXDB_PASSWORD=password"
    ],
    "filters": [],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a handler with PATCH

The `/handlers/:handler` API endpoint provides HTTP PATCH access to update `:handler` definitions, specified by handler name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/handlers/:handler` API endpoint to update the filters array for the `influx-db` handler, resulting in an `HTTP/1.1 200 OK` response and the updated handler definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "filters": [
    "us-west",
    "is_incident"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/influx-db
```

API Specification

/handlers/:handler (PATCH)	
description	Updates the specified Sensu handler.
example URL	http://hostname:8080/api/core/v2/namespaces/default/handlers/influx-db
payload	`{ "filters": [ "us-west", "is_incident" ] }`
response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a handler

The `/handlers/:handler` API endpoint provides HTTP DELETE access to delete a handler from Sensu (specified by the handler name).

Example

The following example shows a request to the `/handlers/:handler` API endpoint to delete the handler `slack`, which will result in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/slack \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/handlers/:handler (DELETE)	
description	Removes the specified handler from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers/slack
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of handlers with response filtering

The `/handlers` API endpoint supports response filtering for a subset of handler data based on labels and the following fields:

- `handler.name`
- `handler.namespace`

- `handler.filters`
- `handler.handlers`
- `handler.mutator`
- `handler.type`

Example

The following example demonstrates a request to the `/handlers` API endpoint with response filtering for only handler definitions in the `default` namespace **and** whose filters include `state_change_only`:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/handlers -G \
--data-urlencode 'fieldSelector=state_change_only in handler.filters &&
handler.namespace == default'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only handler definitions in the `default` namespace **and** whose filters include `state_change_only`:

```
[
  {
    "metadata": {
      "name": "slack",
      "namespace": "default",
      "created_by": "admin"
    },
    "type": "pipe",
    "command": "sensu-slack-handler --channel '#monitoring'",
    "timeout": 0,
    "handlers": null,
    "filters": [
      "state_change_only"
    ],
    "env_vars": null,
    "runtime_assets": [
      "sensu-slack-handler"
```

```
    ],
    "secrets": [
      {
        "name": "SLACK_WEBHOOK_URL",
        "secret": "slack_webhook_url"
      }
    ]
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/handlers (GET) with response filters	
description	Returns the list of handlers that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/handlers
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "metadata": { "name": "slack", "namespace": "default", "created_by": "admin"`

```
    },
    "type": "pipe",
    "command": "sensu-slack-handler --
channel '#monitoring'",
    "timeout": 0,
    "handlers": null,
    "filters": [
      "state_change_only"
    ],
    "env_vars": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": [
      {
        "name": "SLACK_WEBHOOK_URL",
        "secret": "slack_webhook_url"
      }
    ]
  }
]
```

core/v2/hooks

***NOTE**: Requests to `core/v2/hooks` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all hooks

The `/hooks` API endpoint provides HTTP GET access to hook data.

Example

The following example demonstrates a GET request to the `/hooks` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the hook definitions in the `default` namespace:


```
[
  {
    "metadata": {
      "name": "nginx-log",
      "namespace": "default",
      "created_by": "admin"
    },
    "command": "tail -n 100 /var/log/nginx/error.log",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
```

```
  },
  {
    "metadata": {
      "name": "process-tree",
      "namespace": "default",
      "created_by": "admin"
    },
    "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu | head -n 6",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  }
]
```

API Specification

/hooks (GET)	
description	Returns the list of hooks.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "nginx-log",
      "namespace": "default",
```

```
      "created_by": "admin"
    },
    "command": "tail -n 100 /var/log/nginx/error.log",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  },
  {
    "metadata": {
      "name": "process-tree",
      "namespace": "default",
      "created_by": "admin"
    },
    "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu |
head -n 6",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  }
]
```

Create a new hook

The `/hooks` API endpoint provides HTTP POST access to create a hook.

Example

In the following example, an HTTP POST request is submitted to the `/hooks` API endpoint to create the hook `process-tree`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
```

```
    "labels": null,
    "annotations": null
  },
  "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu | head -n 6",
  "timeout": 10,
  "stdin": false
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/hooks (POST)	
description	Creates a Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks
payload	{ "metadata": { "name": "process-tree", "namespace": "default", "labels": null, "annotations": null }, "command": "ps aux", "timeout": 10, "stdin": false }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific hook

The `/hooks/:hook` API endpoint provides HTTP GET access to hook data for specific `:hook` definitions, by hook name.

Example

The following example queries the `/hooks/:hook` API endpoint for the `:hook` named `process-tree`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/process-tree \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:hook` definition (in this example, `process-tree`):

```
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "command": "ps aux",
  "timeout": 10,
  "stdin": false
}
```

API Specification

/hooks/:hook (GET)

description	Returns the specified hook.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "metadata": { "name": "process-tree", "namespace": "default", "created_by": "admin", "labels": null, "annotations": null }, "command": "ps aux", "timeout": 10, "stdin": false }

Create or update a hook

The `/hooks/:hook` API endpoint provides HTTP PUT access to create or update specific `:hook` definitions, by hook name.

Example

In the following example, an HTTP PUT request is submitted to the `/hooks/:hook` API endpoint to create the hook `nginx-log`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "nginx-log",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "tail -n 100 /var/log/nginx/error.log",
  "timeout": 10,
  "stdin": false
  }' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/nginx-log
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/hooks/:hook (PUT)	
description	Creates or updates the specified Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks/nginx-log
payload	{ "metadata": { "name": "nginx-log", "namespace": "default", "labels": null, "annotations": null }, "command": "tail -n 100 /var/log/nginx/error.log", "timeout": 10, "stdin": false }

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a hook with PATCH

The `/hooks/:hook` API endpoint provides HTTP PATCH access to update `:hook` definitions, specified by hook name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/hooks/:hook` API endpoint to update the timeout for the `process-tree` hook, resulting in an `HTTP/1.1 200 OK` response and the updated hook definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "timeout": 20
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hook/process-tree
```

API Specification

/hooks/:hook (PATCH)	
description	Updates the specified Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree
payload	{ "timeout": 20 }
response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a hook

The `/hooks/:hook` API endpoint provides HTTP DELETE access to delete a check hook from Sensu (specified by the hook name).

Example

The following example shows a request to the `/hooks/:hook` API endpoint to delete the hook `process-tree`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/process-tree \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/hooks/:hook (DELETE)	
description	Removes the specified hook from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of hooks with response filtering

The `/hooks` API endpoint supports response filtering for a subset of hook data based on labels and the following fields:

- `hook.name`
- `hook.namespace`

Example

The following example demonstrates a request to the `/hooks` API endpoint with response filtering for only hook definitions in the `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/hooks
-G \
--data-urlencode 'fieldSelector=hook.namespace == production'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only hook definitions in the `production` namespace:

```
[
  {
    "metadata": {
      "name": "process_tree",
      "namespace": "production",
      "created_by": "admin"
    },
    "command": "ps aux",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  },
  {
    "metadata": {
      "name": "restart_nginx",
      "namespace": "production",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "command": "sudo systemctl start nginx",
    "timeout": 60,
    "stdin": false,
    "runtime_assets": null
  }
]
```


***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/hooks (GET) with response filters	
description	Returns the list of hooks that match the response filters applied in the API request.

example url	http://hostname:8080/api/core/v2/hooks
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "process_tree",
      "namespace": "production",
      "created_by": "admin"
    },
    "command": "ps aux",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  },
  {
    "metadata": {
      "name": "restart_nginx",
      "namespace": "production",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "command": "sudo systemctl start nginx",
    "timeout": 60,
    "stdin": false,
    "runtime_assets": null
  }
]
```

core/v2/mutators

***NOTE**: Requests to `core/v2/mutators` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all mutators

The `/mutators` API endpoint provides HTTP GET access to mutator data.

Example

The following example demonstrates a GET request to the `/mutators` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the mutator definitions in the `default` namespace:

```
[
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
```

```
    "env_vars": [],
    "runtime_assets": [],
    "secrets": null,
    "type": "pipe"
  }
]
```

API Specification

/mutators (GET)	
description	Returns the list of mutators.
example url	http://hostname:8080/api/core/v2/namespaces/default/mutators
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
```

```
    "runtime_assets": [],
    "secrets": null,
    "type": "pipe"
  }
]
```

Create a new mutator

The `/mutators` API endpoint provides HTTP POST access to create mutators.

Example

In the following example, an HTTP POST request is submitted to the `/mutators` API endpoint to create the mutator `example-mutator`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": [],
  "secrets": null,
  "type": "pipe"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/mutators (POST)	
description	Creates a Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators
payload
```
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": [],
  "secrets": null,
  "type": "pipe"
}
```
 |
| response codes | ¬ **Success**: 201 (Created)
¬ **Malformed**: 400 (Bad Request)
¬ **Error**: 500 (Internal Server Error) |

Get a specific mutator

The `/mutators/:mutator` API endpoint provides HTTP GET access to mutator data for specific `:mutator` definitions, by mutator name.

Example

The following example queries the `/mutators/:mutator` API endpoint for the `:mutator` named `example-mutator`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:mutator` definition (in this example, `example-mutator`):

```
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": [],
  "secrets": null,
  "type": "pipe"
}
```

API Specification

/mutators/:mutator (GET)	
description	Returns the specified mutator.
example url	http://hostname:8080/api/core/v2/namespaces/default/mutators/example-mutator
response type	Map

response codes

- **Success**: 200 (OK)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

output


```
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": [],
  "secrets": null,
  "type": "pipe"
}
```


Create or update a mutator

The `/mutators/:mutator` API endpoint provides HTTP PUT access to mutator data to create or update specific `:mutator` definitions, by mutator name.

Example

In the following example, an HTTP PUT request is submitted to the `/mutators/:mutator` API endpoint to create the mutator `example-mutator`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
```

```
-d '{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": [],
  "secrets": null,
  "type": "pipe"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/mutators/:mutator (PUT)	
description	Creates or updates a Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators/example-mutator
payload	{ "metadata": { "name": "example-mutator", "namespace": "default", "labels": null, "annotations": null }, "command": "example_mutator.go", "timeout": 0, "env_vars": [], "runtime_assets": [], "secrets": null,

```
  "type": "pipe"
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a mutator with PATCH

The `/mutators/:mutator` API endpoint provides HTTP PATCH access to update `:mutator` definitions, specified by mutator name.

> ***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/mutators/:mutator` API endpoint to update the timeout for the `example-mutator` mutator, resulting in an `HTTP/1.1 200 OK` response and the updated mutator definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "timeout": 10
}' \
```

```
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator
```

API Specification

/mutators/:mutator (PATCH)	
description	Updates the specified Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators/process-tree
payload	{ "timeout": 10 }
response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a mutator

The `/mutators/:mutator` API endpoint provides HTTP DELETE access to delete a mutator from Sensu (specified by the mutator name).

Example

The following example shows a request to the `/mutators/:mutator` API endpoint to delete the mutator `example-mutator`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator \
```

```
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/mutators/:mutator (DELETE)	
description	Removes the specified mutator from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/mutators/example-mutator
response codes	┐ **Success**: 204 (No Content) ┐ **Missing**: 404 (Not Found) ┐ **Error**: 500 (Internal Server Error)

Get a subset of mutators with response filtering

The `/mutators` API endpoint supports response filtering for a subset of mutator data based on labels and the following fields:

- `mutator.name`
- `mutator.namespace`
- `mutator.runtime_assets`

Example

The following example demonstrates a request to the `/mutators` API endpoint with response filtering for only mutator definitions that are in the `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/mutators -G \
--data-urlencode 'fieldSelector=mutator.namespace == production'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only mutator definitions in the `production` namespace:

```
[
  {
    "metadata": {
      "name": "add_check_label",
      "namespace": "production",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "timeout": 0,
    "env_vars": null,
    "runtime_assets": null,
    "secrets": null,
    "type": "javascript",
    "eval": "data = JSON.parse(JSON.stringify(event)); delete
data.check.metadata.name; delete data.entity.metadata.labels.app_id; return
JSON.stringify(data)"
  },
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "production",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": null,
    "runtime_assets": [
      "example-mutator-asset"
    ],
    "secrets": null,
    "type": "pipe"
  }
```

```
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/mutators (GET) with response filters	
description	Returns the list of mutators that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/mutators
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "metadata": { "name": "add_check_label", "namespace": "production", "labels": { "sensu.io/managed_by": "sensuctl" }, "created_by": "admin" }, "timeout": 0, "env_vars": null, "runtime_assets": null, "secrets": null,

```
    "type": "javascript",
    "eval": "data =
JSON.parse(JSON.stringify(event)); delete
data.check.metadata.name; delete
data.entity.metadata.labels.app_id; return
JSON.stringify(data)"
  },
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "production",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": null,
    "runtime_assets": [
      "example-mutator-asset"
    ],
    "secrets": null,
    "type": "pipe"
  }
]
```

core/v2/namespaces

***NOTE**: Requests to `core/v2/namespaces` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all namespaces

The `/namespaces` API endpoint provides HTTP GET access to namespace data.

Example

The following example demonstrates a GET request to the `/namespaces` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains namespace definitions:

```
[
  {
    "name": "default"
  },
  {
    "name": "development"
  }
]
```

API Specification

/namespaces (GET)	
description	Returns the list of namespaces.
example url	http://hostname:8080/api/core/v2/namespaces
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	<pre>[{ "name": "default" }, { "name": "development" }]</pre>

Create a new namespace

The `/namespaces` API endpoint provides HTTP POST access to create Sensu namespaces.

Example

In the following example, an HTTP POST request is submitted to the `/namespaces` API endpoint to

create the namespace `development`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/namespaces (POST)	
description	Creates a Sensu namespace.
example URL	http://hostname:8080/api/core/v2/namespaces
payload	<pre>{ "name": "development" }</pre>
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Create or update a namespace

The `/namespaces/:namespace` API endpoint provides HTTP PUT access to create or update specific Sensu namespaces, by namespace name.

Example

In the following example, an HTTP PUT request is submitted to the `/namespaces/:namespace` API endpoint to create the namespace `development`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/development
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/namespaces/:namespace (PUT)	
description	Creates or updates a Sensu namespace.
example URL	http://hostname:8080/api/core/v2/namespaces/development
payload	`{ "name": "development" }`
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a namespace

The `/namespaces/:namespace` API endpoint provides HTTP DELETE access to delete a namespace from Sensu (specified by the namespace name).

Example

The following example shows a request to the `/namespaces/:namespace` API endpoint to delete the namespace `development`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/development \
-H "Authorization: Key $SENSU_API_KEY"
```

Namespaces must be empty before you can delete them. If the response to your delete request includes `Error: resource is invalid: namespace is not empty`, the namespace may still contain events or other resources. To remove all resources and events so that you can delete a namespace, use this sensuctl dump command (replace `<namespace-name>` with the namespace you want to empty):

```
sensuctl dump entities,events,assets,checks,filters,handlers,secrets/v1.Secret --
namespace <namespace-name> | sensuctl delete
```

API Specification

/namespaces/:namespace (DELETE)	
description	Removes the specified namespace from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/development
response codes	┐ **Success**: 204 (No Content)

- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get a subset of namespaces with response filtering

The `/namespaces` API endpoint supports response filtering for a subset of namespace data based on labels and the field `namespace.name`.

Example

The following example demonstrates a request to the `/namespaces` API endpoint with response filtering for only the namespace definitions for the `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/namespaces -G \
--data-urlencode 'fieldSelector=namespace.name == production'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only namespace definitions for the `production` namespace:

```
[
  {
    "name": "production"
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/namespaces (GET) with response filters	
description	Returns the list of namespaces that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/namespaces
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "name": "production" } ]`

Get all namespaces for a specific user

***COMMERCIAL FEATURE**: Access the `/user-namespaces` API endpoint in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

The `/user-namespaces` API endpoint provides HTTP GET access to the namespaces the current user can access.

Example

The following example demonstrates a GET request to the `/user-namespaces` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/user-namespaces \
-H "Authorization: Key $SENSU_API_KEY"
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only the namespaces the current user can access:

```
[
  {
    "name": "default"
  },
  {
    "name": "development"
  }
]
```

API Specification

/user-namespaces (GET)	
description	Returns the list of namespaces a user has access to.
example url	http://hostname:8080/api/enterprise/user-namespaces
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "name": "default" }, { "name": "development"`

```
    }
]
```

core/v2/pipelines

***IMPORTANT**: The pipelines you can create and manage with this `core/v2/pipelines` API are observation event processing workflows made up of filters, mutators, and handlers.*

*Pipelines are different from the resources you can create and manage with the `enterprise/pipeline/v1` API, which allows you to create and manage resources that can **only** be used in pipelines.*

***NOTE**: Requests to `core/v2/pipelines` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all pipelines

The `/pipelines` API endpoint provides HTTP GET access to pipeline data.

Example

The following example demonstrates a GET request to the `/pipelines` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/pipelines \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the pipeline definitions in the `default` namespace:

```
[
  {
    "metadata": {
```

```
      "name": "labeled_emails",
      "namespace": "default",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "default",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  },
  {
    "metadata": {
      "name": "slack_pipeline",
      "namespace": "default",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "default",
        "filters": [
```

```
            {
              "name": "is_incident",
              "type": "EventFilter",
              "api_version": "core/v2"
            },
            {
              "name": "state_change_only",
              "type": "EventFilter",
              "api_version": "core/v2"
            }
          ],
          "mutator": {
            "name": "add_labels",
            "type": "Mutator",
            "api_version": "core/v2"
          },
          "handler": {
            "name": "slack",
            "type": "Handler",
            "api_version": "core/v2"
          }
        }
      ]
    }
]
```

API Specification

/pipelines (GET)	
description	Returns the list of pipelines.
example url	http://hostname:8080/api/core/v2/namespaces/default/pipelines
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array

response codes

- **Success**: 200 (OK)
- **Error**: 500 (Internal Server Error)

---

output


```
[
  {
    "metadata": {
      "name": "labeled_emails",
      "namespace": "default",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "default",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  },
  {
```

```
    "metadata": {
      "name": "slack_pipeline",
      "namespace": "default",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "default",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "slack",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  }
]
```

Create a new pipeline

The `/pipelines` API endpoint provides HTTP POST access to create a pipeline.

Example

In the following example, an HTTP POST request is submitted to the `/pipelines` API endpoint to create the pipeline resource `labeled_emails`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "labeled_emails",
    "namespace": "default"
  },
  "workflows": [
    {
      "name": "default",
      "filters": [
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "is_incident"
        },
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "state_change_only"
        }
      ],
      "mutator": {
        "api_version": "core/v2",
        "type": "Mutator",
        "name": "add_labels"
      },
      "handler": {
        "api_version": "core/v2",
        "type": "Handler",
        "name": "email"
      }
    }
  ]
```

```
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/pipelines
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/pipelines (POST)	
description	Creates a Sensu pipeline.
example URL	http://hostname:8080/api/core/v2/namespaces/default/pipelines
payload	

```
{
  "metadata": {
    "name": "labeled_email",
    "namespace": "default"
  },
  "workflows": [
    {
      "name": "default",
      "filters": [
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "is_incident"
        },
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "state_change_only"
        }
      ],
      "mutator": {
        "api_version": "core/v2",
        "type": "Mutator",
        "name": "add_labels"
      },
      "handler": {
```

```
        "api_version": "core/v2",
        "type": "Handler",
        "name": "email"
      }
    }
  ]
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific pipeline

The `/pipelines/:pipeline` API endpoint provides HTTP GET access to pipeline data for specific `:pipeline` definitions, by pipeline name.

Example

The following example queries the `/pipelines/:pipeline` API endpoint for the `:pipeline` named `labeled_emails`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/pipelines/labeled_emails \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:pipeline` definition (in this example, `labeled_emails`):

```
{
  "metadata": {
    "name": "labeled_emails",
```

```
    "namespace": "default",
    "created_by": "admin"
  },
  "workflows": [
    {
      "name": "default",
      "filters": [
        {
          "name": "is_incident",
          "type": "EventFilter",
          "api_version": "core/v2"
        },
        {
          "name": "state_change_only",
          "type": "EventFilter",
          "api_version": "core/v2"
        }
      ],
      "mutator": {
        "name": "add_labels",
        "type": "Mutator",
        "api_version": "core/v2"
      },
      "handler": {
        "name": "email",
        "type": "Handler",
        "api_version": "core/v2"
      }
    }
  ]
}
```

API Specification

/pipelines/:pipeline (GET)	
description	Returns a pipeline.
example url	http://hostname:8080/api/core/v2/namespaces/default/pipelines/labeled_emails

response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	


```
{
  "metadata": {
    "name": "labeled_emails",
    "namespace": "default",
    "created_by": "admin"
  },
  "workflows": [
    {
      "name": "default",
      "filters": [
        {
          "name": "is_incident",
          "type": "EventFilter",
          "api_version": "core/v2"
        },
        {
          "name": "state_change_only",
          "type": "EventFilter",
          "api_version": "core/v2"
        }
      ],
      "mutator": {
        "name": "add_labels",
        "type": "Mutator",
        "api_version": "core/v2"
      },
      "handler": {
        "name": "email",
        "type": "Handler",
        "api_version": "core/v2"
      }
    }
```

```
    ]
  }
```

Create or update a pipeline

The `/pipelines/:pipeline` API endpoint provides HTTP PUT access to create or update a specific `:pipeline` definition, by pipeline name.

Example

In the following example, an HTTP PUT request is submitted to the `/pipelines/:pipeline` API endpoint to update `slack_pipeline` to use `javascript_mutator` instead of the `add_labels` mutator:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "slack_pipeline",
    "namespace": "default"
  },
  "workflows": [
    {
      "name": "default",
      "filters": [
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "is_incident"
        },
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "state_change_only"
        }
      ],
```

```
      "mutator": {
        "api_version": "core/v2",
        "type": "Mutator",
        "name": "javascript_mutator"
      },
      "handler": {
        "api_version": "core/v2",
        "type": "Handler",
        "name": "slack"
      }
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/pipelines/slack_pipeline
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/pipelines/:pipeline (PUT)	
description	Creates or updates the specified Sensu pipeline.
example URL	http://hostname:8080/api/core/v2/namespaces/default/pipelines/slack_pipeline
payload	{ "metadata": { "name": "slack_pipeline", "namespace": "default" }, "workflows": [{ "name": "default", "filters": [{ "api_version": "core/v2", "type": "EventFilter", "name": "is_incident"

```
        },
        {
          "api_version": "core/v2",
          "type": "EventFilter",
          "name": "state_change_only"
        }
      ],
      "mutator": {
        "api_version": "core/v2",
        "type": "Mutator",
        "name": "javascript_mutator"
      },
      "handler": {
        "api_version": "core/v2",
        "type": "Handler",
        "name": "slack"
      }
    }
  ]
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Update a pipeline with PATCH

The `/pipelines/:pipeline` API endpoint provides HTTP PATCH access to update `:pipeline` definitions, specified by pipeline name.

***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*

Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/pipelines/:pipeline` API endpoint to update the mutator for `slack_pipeline`, resulting in an `HTTP/1.1 200 OK` response and the updated pipeline definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "workflows": [
    {
      "mutator": {
        "api_version": "core/v2",
        "type": "Mutator",
        "name": "javascript_mutator_2"
      }
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/pipelines/slack_pipeline
```

API Specification

/pipelines/:pipeline (PATCH)	
description	Updates the specified Sensu pipeline.
example URL	http://hostname:8080/api/core/v2/namespaces/default/pipelines/slack_pipeline
payload	`{ "workflows": [`

```
    {
      "mutator": {
        "api_version": "core/v2",
        "type": "Mutator",
        "name": "javascript_mutator"
      }
    }
  ]
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a pipeline

The `/pipelines/:pipeline` API endpoint provides HTTP DELETE access to delete a pipeline from Sensu (specified by the pipeline name).

Example

The following example shows a request to the `/pipelines/:pipeline` API endpoint to delete `slack_pipeline`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/pipelines/slack_pipeline \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/pipelines/:pipeline (DELETE)

description	Removes the specified pipeline from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/pipelines/slack_pipeline
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of pipelines with response filtering

The `/pipelines` API endpoint supports response filtering for a subset of pipeline data based on labels and the following fields:

- `pipeline.name`
- `pipeline.namespace`

Example

The following example demonstrates a request to the `/pipelines` API endpoint with response filtering for only pipeline definitions in the `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/pipelines -G \
--data-urlencode 'fieldSelector=pipeline.namespace == production'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only pipeline definitions in the `production` namespace:

```
[
  {
    "metadata": {
      "name": "sensu_email_alerts",
```

```
      "namespace": "production",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "labeled_email_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  },
  {
    "metadata": {
      "name": "sensu_to_sumo",
      "namespace": "production",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "logs_to_sumologic",
        "handler": {
          "name": "sumologic",
```

```
            "type": "Handler",
            "api_version": "core/v2"
          }
        }
      ]
    }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/pipelines (GET) with response filters	
description	Returns the list of pipelines that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/pipelines
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "metadata": { "name": "sensu_email_alerts", "namespace": "production", "created_by": "admin" }, "workflows": [

```
      {
        "name": "labeled_email_alerts",
        "filters": [
          {
            "name": "is_incident",
            "type": "EventFilter",
            "api_version": "core/v2"
          },
          {
            "name": "state_change_only",
            "type": "EventFilter",
            "api_version": "core/v2"
          }
        ],
        "mutator": {
          "name": "add_labels",
          "type": "Mutator",
          "api_version": "core/v2"
        },
        "handler": {
          "name": "email",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
    ]
  },
  {
    "metadata": {
      "name": "sensu_to_sumo",
      "namespace": "production",
      "created_by": "admin"
    },
    "workflows": [
      {
        "name": "logs_to_sumologic",
        "handler": {
          "name": "sumologic",
          "type": "Handler",
          "api_version": "core/v2"
        }
      }
```

```
        ]
      }
  ]
```

core/v2/rolebindings

***NOTE**: Requests to `core/v2/rolebindings` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all role bindings

The `/rolebindings` API endpoint provides HTTP GET access to role binding data.

Example

The following example demonstrates a GET request to the `/rolebindings` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the role binding definitions in the `default` namespace:

```
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "readers"
      }
    ],
    "role_ref": {
      "type": "Role",
      "name": "read-only"
```

```
    },
    "metadata": {
      "name": "readers-group-binding",
      "namespace": "default",
      "created_by": "admin"
    }
  }
]
```

API Specification

/rolebindings (GET)	
description	Returns the list of role bindings.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "subjects": [ { "type": "Group", "name": "readers" } ], "role_ref": { "type": "Role",`

```
      "name": "read-only"
    },
    "metadata": {
      "name": "readers-group-binding",
      "namespace": "default",
      "created_by": "admin"
    }
  }
]
```

Create a new role binding

The `/rolebindings` API endpoint provides HTTP POST access to create Sensu role bindings.

Example

In the following example, an HTTP POST request is submitted to the `/rolebindings` API endpoint to create a role binding named `readers-group-binding`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default"
  }
```

```
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/rolebindings (POST)	
description	Creates a Sensu role binding.
example URL	http://hostname:8080/api/core/v2/namespaces/default/rolebindings
payload	{ "subjects": [{ "type": "Group", "name": "readers" }], "role_ref": { "type": "Role", "name": "read-only" }, "metadata": { "name": "readers-group-binding", "namespace": "default" } }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific role binding

The `/rolebindings/:rolebinding` API endpoint provides HTTP GET access to role binding data for specific `:rolebinding` definitions, by role binding `name`.

Example

The following example queries the `/rolebindings/:rolebinding` API endpoint for the `:rolebinding` named `readers-group-binding`).

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/readers-group-binding \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:rolebinding` definition (in this example, `readers-group-binding`):

```
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

API Specification

/rolebindings/:rolebinding (GET)	
description	Returns the specified role binding.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/readers-group-binding
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "subjects": [{ "type": "Group", "name": "readers" }], "role_ref": { "type": "Role", "name": "read-only" }, "metadata": { "name": "readers-group-binding", "namespace": "default", "created_by": "admin" } }

Create or update a role binding

The `/rolebindings/:rolebinding` API endpoint provides HTTP PUT access to create or update role binding data for specific `:rolebinding` definitions, by role binding `name`.

Example

In the following example, an HTTP PUT request is submitted to the `/rolebindings/:rolebinding` API endpoint to create the role binding `dev-binding`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "devs"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "workflow-creator"
  },
  "metadata": {
    "name": "dev-binding",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/rolebindings/:rolebinding (PUT)	
description	Creates or updates a Sensu role binding.
example URL	http://hostname:8080/api/core/v2/namespaces/default/role

bindings/dev-binding	
payload
```
{
  "subjects": [
    {
      "type": "Group",
      "name": "devs"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "workflow-creator"
  },
  "metadata": {
    "name": "dev-binding",
    "namespace": "default"
  }
}
```
 |
| response codes | ¬ **Success**: 201 (Created)
¬ **Malformed**: 400 (Bad Request)
¬ **Error**: 500 (Internal Server Error) |

Update a role binding with PATCH

The `/rolebindings/:rolebinding` API endpoint provides HTTP PATCH access to update `:rolebinding` definitions, specified by role binding name.

***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*

Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/rolebindings/:rolebinding` API endpoint to add a group to the subjects array for the `dev-binding` role binding, resulting in an `HTTP/1.1 200 OK` response and the updated role binding definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "dev_team_1"
    },
    {
      "type": "Group",
      "name": "dev_team_2"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/rolebindings/:rolebinding (PATCH)	
description	Updates the specified Sensu role binding.
example URL	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/dev-binding

payload

```
{
  "subjects": [
    {
      "type": "Group",
      "name": "dev_team_1"
    },
    {
      "type": "Group",
      "name": "dev_team_2"
    }
  ]
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a role binding

The `/rolebindings/:rolebinding` API endpoint provides HTTP DELETE access to delete a role binding from Sensu (specified by the role binding name).

Example

The following example shows a request to the `/rolebindings/:rolebinding` API endpoint to delete the role binding `dev-binding`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/rolebindings/:rolebinding (DELETE)	
description	Removes the specified role binding from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/dev-binding
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of role bindings with response filtering

The `/rolebindings` API endpoint supports response filtering for a subset of role binding data based on labels and the following fields:

- `rolebinding.name`
- `rolebinding.namespace`
- `rolebinding.role_ref.name`
- `rolebinding.role_ref.type`

Example

The following example demonstrates a request to the `/rolebindings` API endpoint with response filtering for only role binding definitions with `event-reader` as the rolebinding.role_ref.name:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/rolebindings -G \
--data-urlencode 'fieldSelector="event-reader" in rolebinding.role_ref.name'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only role binding definitions with `event-reader` as the rolebinding.role_ref.name:


```
[
  {
    "subjects": [
      {
        "type": "User",
        "name": "ann"
      },
      {
        "type": "User",
        "name": "bonita"
      },
      {
        "type": "Group",
        "name": "admins"
      },
      {
        "type": "Group",
        "name": "read-events"
      }
    ],
    "role_ref": {
      "type": "Role",
      "name": "event-reader"
    },
    "metadata": {
      "name": "event-reader-binding",
      "namespace": "default",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    }
  }
]
```


***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/rolebindings (GET) with response filters	
description	Returns the list of role bindings that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/rolebindings
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output
```
[
  {
    "subjects": [
      {
        "type": "User",
        "name": "ann"
      },
      {
        "type": "User",
        "name": "bonita"
      },
      {
        "type": "Group",
        "name": "admins"
      },
      {
        "type": "Group",
        "name": "read-events"
      }
```
 |

```
    ],
    "role_ref": {
      "type": "Role",
      "name": "event-reader"
    },
    "metadata": {
      "name": "event-reader-binding",
      "namespace": "default",
      "labels": {
        "sensu.io/managed_by":
"sensuctl"
      },
      "created_by": "admin"
    }
  }
]
```

core/v2/roles

***NOTE**: Requests to `core/v2/roles` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all roles

The `/roles` API endpoint provides HTTP GET access to role data.

Example

The following example demonstrates a GET request to the `/roles` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the role definitions in the `default` namespace:

```
[
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "events"
        ],
```

```
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "event-reader",
      "namespace": "default",
      :created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "read-only",
      "namespace": "default",
      "created_by": "admin"
    }
  }
]
```

API Specification

/roles (GET)	
description	Returns the list of roles.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.

response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "events"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "event-reader",
      "namespace": "default",
      "created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
```

```
    "metadata": {
      "name": "read-only",
      "namespace": "default",
      "created_by": "admin"
    }
  }
]
```

Create a new role

The `/roles` API endpoint provides HTTP POST access to create Sensu roles.

Example

In the following example, an HTTP POST request is submitted to the `/roles` API endpoint to create a role named `event-reader`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": []
    }
  ],
  "metadata": {
    "name": "event-reader",
    "namespace": "default"
  }
```

```
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/roles (POST)	
description	Creates a Sensu role.
example URL	http://hostname:8080/api/core/v2/namespaces/default/roles
payload	`{ "rules": [ { "verbs": [ "get", "list" ], "resources": [ "events" ], "resource_names": [] } ], "metadata": { "name": "event-reader", "namespace": "default" } }`
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific role

The `/roles/:role` API endpoint provides HTTP GET access to role data for specific `:role` definitions, by role name.

Example

The following example queries the `/roles/:role` API endpoint for the `:role` named `read-only`:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:role` definition (in this example, `read-only`):

```
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

API Specification

/roles/:role (GET)	
description	Returns the specified Sensu role.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles/read-only
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "rules": [{ "verbs": ["read"], "resources": ["*"], "resource_names": null }], "metadata": { "name": "read-only", "namespace": "default", "created_by": "admin" } }

Create or update a role

The `/roles/:role` API endpoint provides HTTP PUT access to create or update specific `:role` definitions, by role name.

Example

In the following example, an HTTP PUT request is submitted to the `/roles/:role` API endpoint to create the role `read-only`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/roles/:role (PUT)

description	Creates or updates the specified Sensu role.
example URL	http://hostname:8080/api/core/v2/namespaces/default/roles/event-reader
payload
```
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}
```
 |
| response codes | ¬ **Success**: 201 (Created)
¬ **Malformed**: 400 (Bad Request)
¬ **Error**: 500 (Internal Server Error) |

Update a role with PATCH

The `/roles/:role` API endpoint provides HTTP PATCH access to update `:role` definitions, specified by role name.

***NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a PUT request instead.*

Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/roles/:role` API endpoint to update the verbs array within the rules array for the `global-event-admin` role, resulting in an `HTTP/1.1 200 OK` response and the updated role definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/event-reader
```

API Specification

/roles/:role (PATCH)	
description	Updates the specified Sensu role.

example URL	http://hostname:8080/api/core/v2/namespaces/default/roles/event-reader
payload	<pre>{ "rules": [{ "verbs": ["get", "list"], "resources": ["events"], "resource_names": null }] }</pre>
response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a role

The `/roles/:role` API endpoint provides HTTP DELETE access to delete a role from Sensu (specified by the role name).

Example

The following example shows a request to the `/roles/:role` API endpoint to delete the role `read-only`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
```

```
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/roles/:role (DELETE)	
description	Removes the specified role from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles/read-only
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of roles with response filtering

The `/roles` API endpoint supports response filtering for a subset of role data based on labels and the following fields:

- `role.name`
- `role.namespace`

Example

The following example demonstrates a request to the `/roles` API endpoint with response filtering for only role definitions that are in the `development` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/roles
-G \
--data-urlencode 'fieldSelector=role.namespace == development'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only role definitions in the `development` namespace:


```
[
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "admin_role",
      "namespace": "development",
      "created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ],
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
```

```
            "filters",
            "handlers",
            "hooks",
            "mutators",
            "pipelines",
            "rolebindings",
            "roles",
            "silenced"
          ],
          "resource_names": null
        }
      ],
      "metadata": {
        "name": "namespaced-resources-all-verbs",
        "namespace": "development",
        "created_by": "admin"
      }
    },
    {
      "rules": [
        {
          "verbs": [
            "get",
            "list"
          ],
          "resources": [
            "events"
          ],
          "resource_names": null
        }
      ],
      "metadata": {
        "name": "system:pipeline",
        "namespace": "development"
      }
    }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/roles (GET) with response filters	
description	Returns the list of roles that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/roles
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "rules": [{ "verbs": ["get", "list", "create", "update", "delete"], "resources": ["*"], "resource_names": null }], "metadata": { "name": "admin_role", "namespace": "development",

```
      "created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ],
        "resources": [
          "assets",
          "checks",
          "entities",
          "events",
          "filters",
          "handlers",
          "hooks",
          "mutators",
          "pipelines",
          "rolebindings",
          "roles",
          "silenced"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "namespaced-resources-all-
verbs",
      "namespace": "development",
      "created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get",
```

```
            "list"
          ],
          "resources": [
            "events"
          ],
          "resource_names": null
        }
      ],
      "metadata": {
        "name": "system:pipeline",
        "namespace": "development"
      }
    }
]
```

core/v2/silenced

***NOTE**: Requests to `core/v2/silenced` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all silences

The `/silenced` API endpoint provides HTTP GET access to silencing entry data.

Example

The following example demonstrates a GET request to the `/silenced` API endpoint:

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the silencing definitions in the `default` namespace:

```
[
  {
    "metadata": {
      "name": "*:http",
      "namespace": "default",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "check": "http",
```

```
    "reason": "Testing",
    "begin": 1605024595,
    "expire_at": 0
  },
  {
    "metadata": {
      "name": "linux:*",
      "namespace": "default",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "subscription": "linux",
    "begin": 1542671205,
    "expire_at": 0
  }
]
```


API Specification

/silenced (GET)	
description	Returns the list of silences.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced
pagination	This endpoint does not support pagination.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "*:http",
      "namespace": "default",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "check": "http",
    "reason": "Testing",
    "begin": 1605024595,
    "expire_at": 0
  },
  {
    "metadata": {
      "name": "linux:*",
      "namespace": "default",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "subscription": "linux",
    "begin": 1542671205,
    "expire_at": 0
  }
]
```


Create a new silence

The `/silenced` API endpoint provides HTTP POST access to create silencing entries.

Example

In the following example, an HTTP POST request is submitted to the `/silenced` API endpoint to

create the silencing entry `linux:check_cpu`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "linux:check_cpu",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced
```

The request will return a successful `HTTP/1.1 201 Created` response.

Here's another example that shows an HTTP POST request to the `/silenced` API endpoint to create the silencing entry `*:http`, which will create a silence for any event with the check name `http`, regardless of the originating entities' subscriptions:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "*:http",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
```

```
  "creator": "admin",
  "check": "http",
  "reason": "Testing"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/silenced (POST)	
description	Creates a Sensu silencing entry.
example URL	http://hostname:8080/api/core/v2/namespaces/default/silenced
payload	{ "metadata": { "name": "linux:check_cpu", "namespace": "default", "labels": null, "annotations": null }, "expire": -1, "expire_on_resolve": false, "creator": "admin", "reason": "reason for silence", "subscription": "linux", "begin": 1542671205 }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific silence

The `/silenced/:silenced` API endpoint provides HTTP GET access to silencing entry data for specific `:silenced` definitions, by silencing entry name.

Example

The following example queries the `/silenced/:silenced` API endpoint for the silencing entry named `linux:check_cpu`:

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check_cpu
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:silenced` definition (in this example, `linux:check_cpu`):

```
{
  "metadata": {
    "name": "linux:check_cpu",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}
```

API Specification

/silenced/:silenced (GET)	
description	Returns the specified silencing entry.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check_cpu
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	`{ "metadata": { "name": "linux:check_cpu", "namespace": "default", "created_by": "admin", "labels": null, "annotations": null }, "expire": -1, "expire_on_resolve": false, "creator": "admin", "reason": "reason for silence", "subscription": "linux", "begin": 1542671205 }`

Create or update a silence

The `/silenced/:silenced` API endpoint provides HTTP PUT access to create or update specific `:silenced` definitions, by silencing entry name.

Example

In the following example, an HTTP PUT request is submitted to the `/silenced/:silenced` API endpoint to create the silencing entry `linux:check-server`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "linux:check-server",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-server
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/silenced/:silenced (PUT)	
description	Creates or updates a Sensu silencing entry.
example URL	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check-server
payload	`{ "metadata": { "name": "linux:check-server",`

```
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}
```


---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a silence

The `/silenced/:silenced` API endpoint provides HTTP DELETE access to delete a silencing entry (specified by the silencing entry name).

Example

In the following example, querying the `/silenced/:silenced` API endpoint to delete the the silencing entry named `linux:check_cpu` results in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check_cpu
```

API Specification

/silenced/:silenced (DELETE)	
description	Removes the specified silencing entry from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check_cpu
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get all silences for a specific subscription

The `/silenced/subscriptions/:subscription` API endpoint provides HTTP GET access to silencing entry data by subscription name.

Example

The following example queries the `silenced/subscriptions/:subscription` API endpoint for silences for the given subscription (in this example, for the `linux` subscription):

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/subscriptions/linux
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only silencing definitions for the `linux` subscription:

```
[
  {
    "metadata": {
      "name": "linux:check_cpu",
      "namespace": "default",
      "created_by": "admin",
```

```
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```

API Specification

/silenced/subscriptions/:subscription (GET)	
description	Returns all silences for the specified subscription.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/subscriptions/linux
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	[{ "metadata": { "name": "linux:check_cpu", "namespace": "default", "created_by": "admin",

```
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```

Get all silences for a specific check

The `/silenced/checks/:check` API endpoint provides HTTP GET access to silencing entry data by check name.

Example

The following example queries the `silenced/checks/:check` API endpoint for silences for the specified check (in this example, for the `check_cpu` check):

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/checks/check_cpu
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only silencing definitions for the `check_cpu` check:

```
[
  {
    "metadata": {
      "name": "linux:check_cpu",
      "namespace": "default",
      "created_by": "admin",
```

```
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "check": "linux",
    "begin": 1542671205
  }
]
```

API Specification

/silenced/checks/:check (GET)	
description	Returns all silences for the specified check.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/checks/check_cpu
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	[{ "metadata": { "name": "linux:check_cpu", "namespace": "default", "created_by": "admin", "labels": null,

```
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "check": "linux",
    "begin": 1542671205
  }
]
```

Get a subset of silences with response filtering

The `/silenced` API endpoint supports response filtering for a subset of silences data based on labels and the following fields:

- `silenced.name`
- `silenced.namespace`
- `silenced.check`
- `silenced.creator`
- `silenced.expire_on_resolve`
- `silenced.subscription`

Example

The following example demonstrates a request to the `/silenced` API endpoint with response filtering for only silencing definitions in the `development` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector="development" in silenced.namespace'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that

contains only silencing definitions in the `development` namespace:

```
[
  {
    "metadata": {
      "name": "linux:*",
      "namespace": "development",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "subscription": "linux",
    "begin": 1644868317,
    "expire_at": 0
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/silenced (GET) with response filters	
description	Returns the list of silences that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/silenced
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output


```
[
  {
    "metadata": {
      "name": "linux:*",
      "namespace": "development",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "subscription": "linux",
    "begin": 1644868317,
    "expire_at": 0
  }
]
```

core/v2/tessen

> ***NOTE**: Requests to `core/v2/tessen` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

The core/v2/tessen API endpoints provide HTTP access to manage Tessen configuration. Access to core/v2/tessen is restricted to the default `admin` user.

Get the active Tessen configuration

The `/tessen` API endpoint provides HTTP GET access to the active Tessen configuration.

Example

The following example demonstrates an HTTP GET request to the `/tessen` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/tessen \
-H "Authorization: Key $SENSU_API_KEY"
```

The request returns an HTTP `200 OK` response and a JSON map that contains the active Tessen configuration, indicating whether Tessen is enabled:

```
{
  "opt_out": false
}
```

API Specification

/tessen (GET)	
description	Returns the active Tessen configuration. An `"opt_out": false` response indicates that Tessen is enabled. An `"opt_out": true` response indicates that Tessen is disabled.
example url	http://hostname:8080/api/core/v2/tessen
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
example output	{ "opt_out": false }

Opt in to or out of Tessen

The `/tessen` API endpoint provides HTTP PUT access to opt in to or opt out of Tessen for unlicensed Sensu instances.

***NOTE**: Tessen is enabled by default on Sensu backends and required for licensed Sensu instances.If you have a licensed instance and want to opt out of Tessen, contact your account manager.*

Example

In the following example, an HTTP PUT request is submitted to the `/tessen` API endpoint to opt in to Tessen using the `opt_out` attribute:

```
curl -X PUT \
```

```
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "opt_out": false
}' \
http://127.0.0.1:8080/api/core/v2/tessen
```

The request returns an HTTP `200 OK` response and the resulting Tessen configuration.

```
{
  "opt_out": false
}
```

API Specification

/tessen (PUT)	
description	Updates the active Tessen configuration for unlicensed Sensu instances. Tessen is enabled by default on Sensu backends and required for licensed Sensu instances.
example url	http://hostname:8080/api/core/v2/tessen
request parameters	Required: `opt_out` (for unlicensed instances, set to `false` to enable Tessen; set to `true` to opt out of Tessen).
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
example output	`{ "opt_out": false }`

core/v2/users

***NOTE**: The `core/v2/users` API endpoints allow you to create and manage user credentials with Sensu's built-in basic authentication. To configure user credentials with an external provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC), use Sensu's enterprise/authentication/v2 API endpoints.*

Requests to `core/v2/users` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.

Get all users

The `/users` API endpoint provides HTTP GET access to user data.

Example

The following example demonstrates a GET request to the `/users` API:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/users \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains all user definitions:

```
[
  {
    "username": "admin",
    "groups": [
      "cluster-admins"
    ],
```

```
    "disabled": false
  },
  {
    "username": "agent",
    "groups": [
      "system:agents"
    ],
    "disabled": false
  }
]
```

API Specification

/users (GET)	
description	Returns the list of users.
example url	http://hostname:8080/api/core/v2/users
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`[ { "username": "admin", "groups": [ "cluster-admins" ], "disabled": false }, `

```
  {
    "username": "agent",
    "groups": [
      "system:agents"
    ],
    "disabled": false
  }
]
```

Create a new user

The `/users` API endpoint provides HTTP POST access to create a user using Sensu's basic authentication provider.

Example

The following example demonstrates a POST request to the `/users` API endpoint to create the user `alice`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "temporary",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/users (POST)	
description	Creates a Sensu user.
example URL	http://hostname:8080/api/core/v2/users
payload parameters	Required: `username` (string), `groups` (array; sets of shared permissions that apply to this user), `password` (string; at least eight characters), and `disabled` (when set to `true`, invalidates user credentials and permissions).
payload	<pre>{ "username": "alice", "groups": ["ops"], "password": "temporary", "disabled": false }</pre>
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific user

The `/users/:user` API endpoint provides HTTP GET access to user data for a specific user by `username`.

Example

The following example queries the `/users/:user` API for the `alice` user:

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/users/alice \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:user` definition (in this example, `alice`):

```
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

API Specification

/users/:user (GET)	
description	Returns the specified user.
example url	http://hostname:8080/api/core/v2/users/alice
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	<pre>{ "username": "alice", "groups": ["ops"],</pre>

```
  "disabled": false
}
```

Create or update a user

The `/users/:user` API endpoint provides HTTP PUT access to create or update user data for a specific user by `username`.

> ***NOTE**: Use the `PUT /users/:user/reset_password` or `PUT /users/:user/password` API endpoints to reset or change the user password, respectively.*

Example

The following example demonstrates a PUT request to the `/users` API endpoint to update the user `alice` (for example, to add the user to the `devel` group):

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops",
    "devel"
  ],
  "password": "password",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users/alice
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/users/:user (PUT)	
description	Creates or updates user data for the specified Sensu user.
example URL	http://hostname:8080/api/core/v2/users/alice
payload	{ "username": "alice", "groups": ["ops", "devel"], "password": "password", "disabled": false }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Disable a user

The `/users/:user` API endpoint provides HTTP DELETE access to disable a specific user by `username`.

Example

In the following example, an HTTP DELETE request is submitted to the `/users/:user` API endpoint to disable the user `alice`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice
```

***NOTE**: This endpoint **disables** but does not delete the user. You can reinstate disabled users.*

API Specification

/users/:user (DELETE)	
description	Disables the specified user.
example url	http://hostname:8080/api/core/v2/users/alice
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Reset a user's password

The `/users/:user/reset_password` API endpoint provides HTTP PUT access to reset a user's password.

***NOTE**: The `/users/:user/reset_password` API endpoint requires explicit `users` permissions. With these permissions, you can use `/users/:user/reset_password` to reset a user's password. This differs from the `/users/:user/password` API endpoint, which allows users to change their own passwords without explicit permissions.*

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/reset_password` API endpoint to reset the password for the user `alice`.

The `password_hash` is the user's new password, hashed via bcrypt. Use `sensuctl user hash-password` to generate the `password_hash`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm"
}' \
http://127.0.0.1:8080/api/core/v2/users/alice/reset_password
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/users/:user/reset_password (PUT)	
description	Resets the password for the specified Sensu user.
example URL	http://hostname:8080/api/core/v2/users/alice/reset_password
payload parameters	Required: ¬ `username` : string; the username for the Sensu user ¬ `password_hash` : string; the new user password, hashed via bcrypt
payload	`{ "username": "alice", "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm" }`
response codes	

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Change your password

The `/users/:user/password` API endpoint provides HTTP PUT access to change your Sensu user password.

> ***NOTE**: The `/users/:user/password` API endpoint allows a user to update their own password, without any permissions. This differs from the `/users/:user/reset_password` API endpoint, which requires explicit `users` permissions to change the user password.*

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/password` API endpoint to update the password for the user `alice`.

The `password` is your current password in cleartext. The `password_hash` is your new password hashed via bcrypt. Use `sensuctl user hash-password` to generate the `password_hash`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "password": "P@ssw0rd!",
  "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm"
}' \
http://127.0.0.1:8080/api/core/v2/users/alice/password
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/users/:user/password (PUT)	
description	Changes the password for the specified Sensu user.
example URL	http://hostname:8080/api/core/v2/users/alice/password
payload parameters	Required: ¬ `username` : string; the username for the Sensu user ¬ `password` : string; the user's current password in cleartext ¬ `password_hash` : string; the user's hashed password via bcrypt
payload	`{ "username": "alice", "password": "P@ssw0rd!", "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm" }`
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Reinstate a disabled user

The `/users/:user/reinstate` API endpoint provides HTTP PUT access to reinstate a disabled user.

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/reinstate` API endpoint to reinstate the disabled user `alice`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
http://127.0.0.1:8080/api/core/v2/users/alice/reinstate
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/users/:user/reinstate (PUT)	
description	Reinstates a disabled user.
example URL	http://hostname:8080/api/core/v2/users/alice/reinstate
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Remove a user from all groups

The `/users/:user/groups` API endpoint provides HTTP DELETE access to remove the specified user from all groups.

Example

In the following example, an HTTP DELETE request is submitted to the `/users/:user/groups` API endpoint to remove the user `alice` from all groups within Sensu, resulting in a successful `HTTP/1.1`

`204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups
```

API Specification

/users/:user/groups (DELETE)	
description	Removes the specified user from all groups.
example url	http://hostname:8080/api/core/v2/users/alice/groups
response codes	┐ **Success**: 204 (No Content) ┐ **Missing**: 404 (Not Found) ┐ **Error**: 500 (Internal Server Error)

Assign a user to a group

The `/users/:user/groups/:group` API endpoint provides HTTP PUT access to assign a user to a group.

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/groups/:group` API endpoint to add the user `alice` to the group `ops`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/users/:user/groups/:group (PUT)	
description	Adds the specified user to the specified group.
example URL	http://hostname:8080/api/core/v2/users/alice/groups/ops
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Remove a user from a specific group

The `/users/:user/groups/:group` API endpoint provides HTTP DELETE access to remove the specified user from a specific group.

Example

In the following example, an HTTP DELETE request is submitted to the `/users/:user/groups/:group` API endpoint to remove the user `alice` from the group `ops`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops
```

API Specification

/users/:user/groups/:group (DELETE)	
description	Removes the specified user from the specified group.
example url	http://hostname:8080/api/core/v2/users/alice/groups/ops
response codes	┐ **Success**: 204 (No Content) ┐ **Missing**: 404 (Not Found) ┐ **Error**: 500 (Internal Server Error)

Get a subset of users with response filtering

The `/users` API endpoint supports response filtering for a subset of user data based on labels and the following fields:

- `user.username`
- `user.disabled`
- `user.groups`

Example

The following example demonstrates a request to the `/users` API endpoint with response filtering for only user definitions whose user.groups include `dev`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/users
-G \
--data-urlencode 'fieldSelector="dev" in user.groups'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only user definitions whose user.groups include `dev`:

```
[
  {
    "username": "alice",
    "groups": [
      "ops",
      "dev"
    ],
    "disabled": false
  },
  {
    "username": "balan",
    "groups": [
      "testing",
      "dev"
    ],
    "disabled": false
  }
]
```


***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/users (GET) with response filters	
description	Returns the list of users that match the response filters applied in the API request.
example url	http://hostname:8080/api/core/v2/users
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output

```
[
  {
    "username": "alice",
    "groups": [
      "ops",
      "dev"
    ],
    "disabled": false
  },
  {
    "username": "balan",
    "groups": [
      "testing",
      "dev"
    ],
    "disabled": false
  }
]
```

Enterprise APIs

***COMMERCIAL FEATURE**: Access Sensu's enterprise APIs in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Sensu's enterprise APIs provide programmatic access to commercial features. The enterprise APIs include:

- enterprise/authentication/v2
- enterprise/bsm/v1
- enterprise/federation/v1
- enterprise/pipeline/v1
- enterprise/prune/v1alpha
- enterprise/searches/v1
- enterprise/secrets/v1
- enterprise/store/v1
- enterprise/web/v1

enterprise/authentication/v2

***COMMERCIAL FEATURE**: Access authentication providers for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/authentication/v2` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get active authentication provider configurations

The `/authproviders` API endpoint provides HTTP GET access to authentication provider configuration in Sensu.

Example

The following example queries the `/authproviders` API endpoint for the authentication provider configurations in Sensu:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/authentication/v2/authproviders \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the authentication provider configurations:

```
[
  {
    "type": "oidc",
    "api_version": "authentication/v2",
    "metadata": {
```

```
      "name": "oidc_auth",
      "created_by": "admin"
    },
    "spec": {
      "additional_scopes": [
        "groups",
        "email"
      ],
      "client_id": "xxxxxxxxxxxxxxxxxxx",
      "client_secret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
      "disable_offline_access": false,
      "groups_claim": "groups",
      "groups_prefix": "oidc:",
      "redirect_uri": "http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/oidc/callback",
      "server": "https://oidc.example.com:9031",
      "username_claim": "email",
      "username_prefix": "oidc:"
    }
  },
  {
    "type": "ldap",
    "api_version": "authentication/v2",
    "metadata": {
      "name": "openldap",
      "created_by": "admin"
    },
    "spec": {
      "groups_prefix": "",
      "servers": [
        {
          "binding": {
            "password": "YOUR_PASSWORD",
            "user_dn": "cn=binder,dc=acme,dc=org"
          },
          "client_cert_file": "",
          "client_key_file": "",
          "default_upn_domain": "",
          "group_search": {
            "attribute": "member",
            "base_dn": "dc=acme,dc=org",
            "name_attribute": "cn",
```

```
            "object_class": "groupOfNames"
          },
          "host": "127.0.0.1",
          "insecure": false,
          "port": 636,
          "security": "tls",
          "trusted_ca_file": "",
          "user_search": {
            "attribute": "uid",
            "base_dn": "dc=acme,dc=org",
            "name_attribute": "cn",
            "object_class": "person"
          }
        }
      ],
      "username_prefix": ""
    }
  }
]
```


API Specification

/authproviders (GET)	
description	Returns the list of active authentication providers.
example url	http://hostname:8080/api/enterprise/authentication/v2/authproviders
query parameters	`types` : Defines which type of authentication provider to retrieve. Join with `&` to retrieve multiple types: `?types=AD&types=OIDC` .
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters. Read the API overview for details.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output


```
[
  {
    "type": "oidc",
    "api_version": "authentication/v2",
    "metadata": {
      "name": "oidc_auth",
      "created_by": "admin"
    },
    "spec": {
      "additional_scopes": [
        "groups",
        "email"
      ],
      "client_id": "xxxxxxxxxxxxxxxxxxx",
      "client_secret":
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
      "disable_offline_access": false,
      "groups_claim": "groups",
      "groups_prefix": "oidc:",
      "redirect_uri": "http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2
/oidc/callback",
      "server": "https://oidc.example.com:9031",
      "username_claim": "email",
      "username_prefix": "oidc:"
    }
  },
  {
    "type": "ldap",
    "api_version": "authentication/v2",
    "metadata": {
      "name": "openldap",
      "created_by": "admin"
    },
    "spec": {
      "groups_prefix": "",
      "servers": [
        {
          "binding": {
            "password": "YOUR_PASSWORD",
            "user_dn": "cn=binder,dc=acme,dc=org"
```

```
            },
            "client_cert_file": "",
            "client_key_file": "",
            "default_upn_domain": "",
            "group_search": {
              "attribute": "member",
              "base_dn": "dc=acme,dc=org",
              "name_attribute": "cn",
              "object_class": "groupOfNames"
            },
            "host": "127.0.0.1",
            "insecure": false,
            "port": 636,
            "security": "tls",
            "trusted_ca_file": "",
            "user_search": {
              "attribute": "uid",
              "base_dn": "dc=acme,dc=org",
              "name_attribute": "cn",
              "object_class": "person"
            }
          }
        ],
        "username_prefix": ""
      }
    }
]
```


Get the configuration for a specific authentication provider

The `/authproviders/:name` API endpoint provides HTTP GET access to the authentication provider configuration for a specific `:name`.

Example

In the following example, an HTTP GET request is submitted to the `/authproviders/:name` API endpoint to retrieve the `openldap` authenthication provider configuration:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/authentication/v2/authproviders/openldap \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json'
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested authentication provider `:name` definition (in this example, `openldap`):


```
{
  "type": "ldap",
  "api_version": "authentication/v2",
  "metadata": {
    "name": "openldap",
    "created_by": "admin"
  },
  "spec": {
    "groups_prefix": "",
    "servers": [
      {
        "binding": {
          "password": "YOUR_PASSWORD",
          "user_dn": "cn=binder,dc=acme,dc=org"
        },
        "client_cert_file": "",
        "client_key_file": "",
        "default_upn_domain": "",
        "group_search": {
          "attribute": "member",
          "base_dn": "dc=acme,dc=org",
          "name_attribute": "cn",
          "object_class": "groupOfNames"
        },
        "host": "127.0.0.1",
        "insecure": false,
        "port": 636,
        "security": "tls",
        "trusted_ca_file": "",
        "user_search": {
          "attribute": "uid",
```

```
            "base_dn": "dc=acme,dc=org",
            "name_attribute": "cn",
            "object_class": "person"
          }
        }
      ],
      "username_prefix": ""
    }
}
```

API Specification

/authproviders/:name (GET)	
description	Returns the configuration for an authentication provider for the specified configured provider name.
example url	http://hostname:8080/api/enterprise/authentication/v2/authproviders/openldap
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "ldap", "api_version": "authentication/v2", "metadata": { "name": "openldap", "created_by": "admin" }, "spec": { "groups_prefix": "", "servers": [{

```
        "binding": {
          "password": "YOUR_PASSWORD",
          "user_dn": "cn=binder,dc=acme,dc=org"
        },
        "client_cert_file": "",
        "client_key_file": "",
        "default_upn_domain": "",
        "group_search": {
          "attribute": "member",
          "base_dn": "dc=acme,dc=org",
          "name_attribute": "cn",
          "object_class": "groupOfNames"
        },
        "host": "127.0.0.1",
        "insecure": false,
        "port": 636,
        "security": "tls",
        "trusted_ca_file": "",
        "user_search": {
          "attribute": "uid",
          "base_dn": "dc=acme,dc=org",
          "name_attribute": "cn",
          "object_class": "person"
        }
      }
    ],
    "username_prefix": ""
  }
}
```

Create or update the configuration for a specific authentication provider

The `/authproviders/:name` API endpoint provides HTTP PUT access to create or update the authentication provider configuration for a specific `:name`.

Example

In the following example, an HTTP PUT request is submitted to the `/authproviders/:name` API endpoint to create the `openldap` authenthication provider:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "Type": "ldap",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "binding": {
          "user_dn": "cn=binder,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "openldap"
  }
}' \
http://127.0.0.1:8080/api/enterprise/authentication/v2/authproviders/openldap
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/authproviders/:name (PUT)

description	Creates or updates the authentication provider configuration for the specified name. Read the authentication guide for more information about supported providers.
example url	http://hostname:8080/api/enterprise/authentication/v2/authproviders/openldap
payload	<pre>{ "Type": "ldap", "api_version": "authentication/v2", "spec": { "servers": [{ "host": "127.0.0.1", "binding": { "user_dn": "cn=binder,dc=acme,dc=org", "password": "YOUR_PASSWORD" }, "group_search": { "base_dn": "dc=acme,dc=org" }, "user_search": { "base_dn": "dc=acme,dc=org" } }] }, "metadata": { "name": "openldap" } }</pre>
payload parameters	All attributes shown in the example payload are required. For more information about configuring authentication providers, read the authentication guide.
response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬

- **Error**: 500 (Internal Server Error)

Delete the configuration for a specific authentication provider

The `/authproviders/:name` API endpoint provides HTTP DELETE access to delete the authentication provider configuration from Sensu for a specific `:name`.

Example

The following example shows a request to the `/authproviders/:name` API endpoint to delete the configuration for the authentication provider `openldap`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/authproviders/openldap
```

API Specification

/authproviders/:name (DELETE)	
description	Deletes the authentication provider configuration from Sensu for the specified name.
example url	http://hostname:8080/api/enterprise/authentication/v2/authproviders/openldap
response codes	- **Success**: 204 (No Content) - **Missing**: 404 (Not Found) - **Error**: 500 (Internal Server Error)

enterprise/bsm/v1

***COMMERCIAL FEATURE**: Access business service monitoring (BSM) in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Business service monitoring (BSM) is in public preview and is subject to change.*

Requests to `enterprise/bsm/v1` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.

Get all service components

The `/service-components` API endpoint provides HTTP GET access to a list of service components.

Example

The following example demonstrates a GET request to the `/service-components` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/service-components \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the service component definitions in the `default` namespace:

```
[
  {
    "type": "ServiceComponent",
    "api_version": "bsm/v1",
    "metadata": {
```

```
      "name": "webservers",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "cron": "",
      "handlers": [
        "slack"
      ],
      "interval": 60,
      "query": [
        {
          "type": "fieldSelector",
          "value": "webserver in event.check.subscriptions"
        }
      ],
      "rules": [
        {
          "arguments": {
            "critical_threshold": 70,
            "warning_threshold": 50
          },
          "name": "webservers_50-70",
          "template": "aggregate"
        }
      ],
      "services": [
        "website-services"
      ]
    }
  }
]
```

API Specification

/service-components (GET)	
description	Returns the list of service components.
example url	http://hostname:8080/api/enterprise/bsm/v1/namespaces/defau

	lt/service-components
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output
```
[
  {
    "type": "ServiceComponent",
    "api_version": "bsm/v1",
    "metadata": {
      "name": "webservers",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "cron": "",
      "handlers": [
        "slack"
      ],
      "interval": 60,
      "query": [
        {
          "type": "fieldSelector",
          "value": "webserver in
event.check.subscriptions"
        }
      ],
      "rules": [
        {
          "arguments": {
            "critical_threshold": 70,
            "warning_threshold": 50
          },
          "name": "webservers_50-70",
          "template": "aggregate"
        }
      ],
```
 |

```
      "services": [
        "website-services"
      ]
    }
  }
]
```

Create a new service component

The `/service-components` API endpoint provides HTTP POST access to create service components.

Example

The following example demonstrates a request to the `/service-components` API endpoint to create the service component `webservers`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "ServiceComponent",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "webservers"
  },
  "spec": {
    "cron": "",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in event.check.subscriptions"
      }
```

```
    ],
    "rules": [
      {
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        },
        "name": "webservers_50-70",
        "template": "aggregate"
      }
    ],
    "services": [
      "website-services"
    ]
  }
}' \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/service-components
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/service-components (POST)	
description	Creates a new business service component (if none exists).
example URL	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/service-components
payload	{ "type": "ServiceComponent", "api_version": "bsm/v1", "metadata": { "name": "webservers" }, "spec": { "cron": "", "handlers": ["slack"

```
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in
event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        },
        "name": "webservers_50-70",
        "template": "aggregate"
      }
    ],
    "services": [
      "website-services"
    ]
  }
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific service component

The `/service-components/:service-component` API endpoint provides HTTP GET access to data for a specific `:service-component`, by service compnent name.

Example

The following example queries the `/service-components/:service-component` API endpoint for a specific `:service-component`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/service-components/webservers \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:service-component` definition (in this example, `webservers`):


```
{
  "type": "ServiceComponent",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "webservers",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "cron": "",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        },
        "name": "webservers_50-70",
        "template": "aggregate"
```

```
      }
    ],
    "services": [
      "website-services"
    ]
  }
}
```

API Specification

/service-components/:service-component (GET)	
description	Returns the specified business service component.
example url	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/service-components/webservers
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "ServiceComponent", "api_version": "bsm/v1", "metadata": { "name": "webservers", "namespace": "default", "created_by": "admin" }, "spec": { "cron": "", "handlers": [

```
      "slack"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in
event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "arguments": {
          "critical_threshold":
70,
          "warning_threshold":
50
        },
        "name": "webservers_50-
70",
        "template": "aggregate"
      }
    ],
    "services": [
      "website-services"
    ]
  }
}
```

Create or update a service component

The `/service-components/:service-component` API endpoint provides HTTP PUT access to create or update a specific `:service-component`, by service component name.

Example

The following example demonstrates a request to the `/service-components/:service-component`

API endpoint to update the service component `webservers`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "ServiceComponent",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "webservers"
  },
  "spec": {
    "cron": "",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "query": [
      {
        "type": "fieldSelector",
        "value": "webserver in event.check.subscriptions"
      }
    ],
    "rules": [
      {
        "arguments": {
          "critical_threshold": 70,
          "warning_threshold": 50
        },
        "name": "webservers_50-70",
        "template": "aggregate"
      }
    ],
    "services": [
      "website-services"
    ]
  }
}' \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/service-components/webservers
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/service-components/:service-component (PUT)	
description	Creates or updates the specified business service component.
example URL	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/service-components/webservers
payload	{ "type": "ServiceComponent", "api_version": "bsm/v1", "metadata": { "name": "webservers" }, "spec": { "cron": "", "handlers": ["slack"], "interval": 60, "query": [{ "type": "fieldSelector", "value": "webserver in event.check.subscriptions" }], "rules": [{ "arguments": { "critical_threshold": 70, "warning_threshold": 50

```
        },
        "name": "webservers_50-
70",
        "template": "aggregate"
      }
    ],
    "services": [
      "website-services"
    ]
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a service component

The `/service-components/:service-component` API endpoint provides HTTP DELETE access to delete the specified service component from Sensu.

Example

The following example shows a request to the `/service-components/:service-component` API endpoint to delete the service component `webservers`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/service-
components/webservers
```

API Specification

/service-components/:service-component (DELETE)	
description	Deletes the specified business service component from Sensu.
example url	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/service-components/webservers
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get all rule templates

The `/rule-templates` API endpoint provides HTTP GET access to a list of rule templates.

Example

The following example demonstrates a GET request to the `/rule-templates` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the rule template definitions in the `default` namespace:

```
[
  {
```

```
    "type": "RuleTemplate",
    "api_version": "bsm/v1",
    "metadata": {
      "name": "aggregate",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "arguments": {
        "properties": {
          "critical_count": {
            "description": "create an event with a critical status if there the
number of critical events is equal to or greater than this count",
            "type": "number"
          },
          "critical_threshold": {
            "description": "create an event with a critical status if the percentage
of non-zero events is equal to or greater than this threshold",
            "type": "number"
          },
          "metric_handlers": {
            "default": {},
            "description": "metric handlers to use for produced metrics",
            "items": {
              "type": "string"
            },
            "type": "array"
          },
          "produce_metrics": {
            "default": {},
            "description": "produce metrics from aggregate data and include them in
the produced event",
            "type": "boolean"
          },
          "set_metric_annotations": {
            "default": {},
            "description": "annotate the produced event with metric annotations",
            "type": "boolean"
          },
          "warning_count": {
            "description": "create an event with a warning status if there the
number of critical events is equal to or greater than this count",
```

```
            "type": "number"
          },
          "warning_threshold": {
            "description": "create an event with a warning status if the percentage
of non-zero events is equal to or greater than this threshold",
            "type": "number"
          }
        },
        "required": null
      },
      "description": "Monitor a distributed service - aggregate one or more events
into a single event. This BSM rule template allows you to treat the results of
multiple disparate check executions - executed across multiple disparate systems -
as a single event. This template is extremely useful in dynamic environments and/or
environments that have a reasonable tolerance for failure. Use this template when a
service can be considered healthy as long as a minimum threshold is satisfied (for
example, at least 5 healthy web servers? at least 70% of N processes healthy?).",
      "eval": "\nif (events && events.length == 0) {\n    event.check.output =
\"WARNING: No events selected for aggregate\n\";\n    event.check.status = 1;\n
return event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_event_count\"] =
events.length;\n\npercentOK = sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var handlers = [];\n\n    if
(!!args[\"metric_handlers\"]) {\n        handlers =
args[\"metric_handlers\"].slice();\n    }\n\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n    event.timestamp = ts;\n\n    var tags = [\n
{\n            name: \"service\",\n            value: event.entity.name\n
},\n        {\n            name: \"entity\",\n            value: event.entity.name\n
},\n        {\n            name: \"check\",\n            value: event.check.name\n
}\n    ];\n\n    event.metrics = sensu.NewMetrics({\n        handlers: handlers,\n
points: [\n            {\n                name: \"percent_non_zero\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n                name: \"percent_ok\",\n
timestamp: ts,\n                value: percentOK,\n                tags: tags\n
},\n            {\n                name: \"percent_warning\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n                name:
\"percent_critical\",\n                timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n                tags: tags\n
},\n            {\n                name: \"percent_unknown\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n                name:
\"count_non_zero\",\n                timestamp: ts,\n                value:
```

```
sensu.CountBySeverity(\"non-zero\"),\n                tags: tags\n            },\n
{\n                name: \"count_ok\",\n                  timestamp: ts,\n
value: sensu.CountBySeverity(\"ok\"),\n                tags: tags\n            },\n
{\n                name: \"count_warning\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"warning\"),\n                tags: tags\n
},\n            {\n                name: \"count_critical\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n                name:
\"count_unknown\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n                tags: tags\n            }\n
]\n    });\n\n    if (!!args[\"set_metric_annotations\"]) {\n        var i = 0;\n\n
while(i < event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n            i++;\n        }\n
}\n}\n\nif (!!args[\"critical_threshold\"] && percentOK <=
args[\"critical_threshold\"]) {\n    event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] && percentOK <=
args[\"warning_threshold\"]) {\n    event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit >= args[\"critical_count\"])
{\n        event.check.output = \"CRITICAL: \" + args[\"critical_count\"].toString()
+ \" or more selected events are in a critical state (\" + crit.toString() +
\")\n\";\n        event.check.status = 2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn = sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n        event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected events are in a warning
state (\" + warn.toString() + \")\n\";\n        event.check.status = 1;\n
return event;\n    }\n}\n\nevent.check.output = \"Everything looks good (\" +
percentOK.toString() + \"% OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
    }
  }
]
```

API Specification

/rule-templates (GET)	
description	Returns the list of rule templates.
example url	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "type": "RuleTemplate",
    "api_version": "bsm/v1",
    "metadata": {
      "name": "aggregate",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "arguments": {
        "properties": {
          "critical_count": {
            "description": "create an event with a
critical status if there the number of critical events
is equal to or greater than this count",
            "type": "number"
          },
          "critical_threshold": {
            "description": "create an event with a
critical status if the percentage of non-zero events is
equal to or greater than this threshold",
            "type": "number"
          },
          "metric_handlers": {
            "default": {},
            "description": "metric handlers to use for
produced metrics",
```

```
            "items": {
              "type": "string"
            },
            "type": "array"
          },
          "produce_metrics": {
            "default": {},
            "description": "produce metrics from
aggregate data and include them in the produced event",
            "type": "boolean"
          },
          "set_metric_annotations": {
            "default": {},
            "description": "annotate the produced event
with metric annotations",
            "type": "boolean"
          },
          "warning_count": {
            "description": "create an event with a
warning status if there the number of critical events is
equal to or greater than this count",
            "type": "number"
          },
          "warning_threshold": {
            "description": "create an event with a
warning status if the percentage of non-zero events is
equal to or greater than this threshold",
            "type": "number"
          }
        },
        "required": null
      },
      "description": "Monitor a distributed service -
aggregate one or more events into a single event. This
BSM rule template allows you to treat the results of
multiple disparate check executions - executed across
multiple disparate systems - as a single event. This
template is extremely useful in dynamic environments
and/or environments that have a reasonable tolerance for
failure. Use this template when a service can be
considered healthy as long as a minimum threshold is
satisfied (for example, at least 5 healthy web servers?
```

```
at least 70% of N processes healthy?).",
      "eval": "\nif (events && events.length == 0) {\n
event.check.output = \"WARNING: No events selected for
aggregate\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_e
vent_count\"] = events.length;\n\npercentOK =
sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var handlers =
[];\n\n    if (!!args[\"metric_handlers\"]) {\n
handlers = args[\"metric_handlers\"].slice();\n    }\n\n
var ts = Math.floor(new Date().getTime() / 1000);\n\n
event.timestamp = ts;\n\n    var tags = [\n        {\n
name: \"service\",\n            value:
event.entity.name\n        },\n        {\n
name: \"entity\",\n            value:
event.entity.name\n        },\n        {\n
name: \"check\",\n            value: event.check.name\n
}\n    ];\n\n    event.metrics = sensu.NewMetrics({\n
handlers: handlers,\n        points: [\n            {\n
name: \"percent_non_zero\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"percent_ok\",\n                timestamp: ts,\n
value: percentOK,\n                tags: tags\n
},\n            {\n                name:
\"percent_warning\",\n                timestamp: ts,\n
value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"percent_critical\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"percent_unknown\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n
name: \"count_non_zero\",\n                timestamp:
ts,\n                value: sensu.CountBySeverity(\"non-
zero\"),\n                tags: tags\n            },\n
{\n                name: \"count_ok\",\n
timestamp: ts,\n                value:
```

```
sensu.CountBySeverity(\"ok\"),\n                tags:
tags\n            },\n            {\n
name: \"count_warning\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"count_critical\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"count_unknown\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n            }\n        ]\n    });\n\n    if
(!!args[\"set_metric_annotations\"]) {\n        var i =
0;\n\n        while(i < event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" +
event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n
i++;\n        }\n    }\n}\n\nif
(!!args[\"critical_threshold\"] && percentOK <=
args[\"critical_threshold\"]) {\n    event.check.output
= \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of
selected events are OK (\" + percentOK.toString() +
\"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] &&
percentOK <= args[\"warning_threshold\"]) {\n
event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected
events are OK (\" + percentOK.toString() + \"%)\n\";\n
event.check.status = 1;\n    return event;\n}\n\nif
(!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit >=
args[\"critical_count\"]) {\n        event.check.output
= \"CRITICAL: \" + args[\"critical_count\"].toString() +
\" or more selected events are in a critical state (\" +
crit.toString() + \")\n\";\n        event.check.status =
2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn =
sensu.CountBySeverity(\"warning\");\n\n    if (warn >=
args[\"warning_count\"]) {\n        event.check.output =
```

```
\"WARNING: \" + args[\"warning_count\"].toString() + \"
or more selected events are in a warning state (\" +
warn.toString() + \")\n\";\n        event.check.status =
1;\n        return event;\n
}\n}\n\nevent.check.output = \"Everything looks good (\"
+ percentOK.toString() + \"% OK)\";\nevent.check.status
= 0;\n\nreturn event;\n"
    }
  }
]
```

Create a new rule template

The `/rule-templates` API endpoint provides HTTP POST access to create rule templates.

Example

The following example demonstrates a request to the `/rule-templates` API endpoint to create the rule template `aggregate`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "RuleTemplate",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "aggregate"
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
```

```
        "critical_threshold": {
          "description": "create an event with a critical status if the percentage
of non-zero events is equal to or greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers to use for produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from aggregate data and include them in
the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a warning status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event with a warning status if the percentage of
non-zero events is equal to or greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service - aggregate one or more events
into a single event. This BSM rule template allows you to treat the results of
multiple disparate check executions - executed across multiple disparate systems -
as a single event. This template is extremely useful in dynamic environments and/or
environments that have a reasonable tolerance for failure. Use this template when a
```

```
service can be considered healthy as long as a minimum threshold is satisfied (for
example, at least 5 healthy web servers? at least 70% of N processes healthy?).",
    "eval": "\nif (events && events.length == 0) {\n    event.check.output =
\"WARNING: No events selected for aggregate\n\";\n    event.check.status = 1;\n
return event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_event_count\"] =
events.length;\n\npercentOK = sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var handlers = [];\n\n    if
(!!args[\"metric_handlers\"]) {\n        handlers =
args[\"metric_handlers\"].slice();\n    }\n\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n    event.timestamp = ts;\n\n    var tags = [\n
{\n            name: \"service\",\n            value: event.entity.name\n
},\n        {\n            name: \"entity\",\n            value: event.entity.name\n
},\n        {\n            name: \"check\",\n            value: event.check.name\n
}\n    ];\n\n    event.metrics = sensu.NewMetrics({\n        handlers: handlers,\n
points: [\n            {\n                name: \"percent_non_zero\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n                name: \"percent_ok\",\n
timestamp: ts,\n                value: percentOK,\n                tags: tags\n
},\n            {\n                name: \"percent_warning\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n                name:
\"percent_critical\",\n                timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n                tags: tags\n
},\n            {\n                name: \"percent_unknown\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n                name:
\"count_non_zero\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"non-zero\"),\n                tags: tags\n            },\n
{\n                name: \"count_ok\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"ok\"),\n                tags: tags\n            },\n
{\n                name: \"count_warning\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"warning\"),\n                tags: tags\n
},\n            {\n                name: \"count_critical\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n                name:
\"count_unknown\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n                tags: tags\n            }\n
]\n    });\n\n    if (!!args[\"set_metric_annotations\"]) {\n        var i = 0;\n\n
while(i < event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n            i++;\n        }\n
}\n}\n\nif (!!args[\"critical_threshold\"] && percentOK <=
```

```
args[\"critical_threshold\"]) {\n    event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] && percentOK <=
args[\"warning_threshold\"]) {\n    event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit >= args[\"critical_count\"])
{\n        event.check.output = \"CRITICAL: \" + args[\"critical_count\"].toString()
+ \" or more selected events are in a critical state (\" + crit.toString() +
\")\n\";\n        event.check.status = 2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn = sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n        event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected events are in a warning
state (\" + warn.toString() + \")\n\";\n        event.check.status = 1;\n
return event;\n    }\n}\n\nevent.check.output = \"Everything looks good (\" +
percentOK.toString() + \"% OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}' \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/rule-templates (POST)	
description	Creates a new rule template (if none exists).
example URL	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates
payload	{ "type": "RuleTemplate", "api_version": "bsm/v1", "metadata": { "name": "aggregate" },

```
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a
critical status if there the number of critical events
is equal to or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a
critical status if the percentage of non-zero events is
equal to or greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers to use for
produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from
aggregate data and include them in the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event
with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a
warning status if there the number of critical events
is equal to or greater than this count",
          "type": "number"
        },
```

```
        "warning_threshold": {
          "description": "create an event with a
warning status if the percentage of non-zero events is
equal to or greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service -
aggregate one or more events into a single event. This
BSM rule template allows you to treat the results of
multiple disparate check executions - executed across
multiple disparate systems - as a single event. This
template is extremely useful in dynamic environments
and/or environments that have a reasonable tolerance
for failure. Use this template when a service can be
considered healthy as long as a minimum threshold is
satisfied (for example, at least 5 healthy web servers?
at least 70% of N processes healthy?).",
    "eval": "\nif (events && events.length == 0) {\n
event.check.output = \"WARNING: No events selected for
aggregate\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_
event_count\"] = events.length;\n\npercentOK =
sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var handlers =
[];\n\n    if (!!args[\"metric_handlers\"]) {\n
handlers = args[\"metric_handlers\"].slice();\n
}\n\n    var ts = Math.floor(new Date().getTime() /
1000);\n\n    event.timestamp = ts;\n\n    var tags =
[\n        {\n            name: \"service\",\n
value: event.entity.name\n        },\n        {\n
name: \"entity\",\n            value:
event.entity.name\n        },\n        {\n
name: \"check\",\n            value: event.check.name\n
}\n    ];\n\n    event.metrics = sensu.NewMetrics({\n
handlers: handlers,\n        points: [\n            {\n
name: \"percent_non_zero\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
```

```
name: \"percent_ok\",\n                timestamp: ts,\n
value: percentOK,\n                tags: tags\n
},\n            {\n                name:
\"percent_warning\",\n                   timestamp: ts,\n
value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"percent_critical\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"percent_unknown\",\n                timestamp:
ts,\n                value:
sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n
name: \"count_non_zero\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"count_ok\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"ok\"),\n
tags: tags\n            },\n            {\n
name: \"count_warning\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"count_critical\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"count_unknown\",\n                timestamp:
ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n            }\n        ]\n    });\n\n    if
(!!args[\"set_metric_annotations\"]) {\n        var i =
0;\n\n        while(i < event.metrics.points.length)
{\n
event.annotations[\"io.sensu.bsm.selected_event_\" +
event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n
i++;\n        }\n    }\n}\n\nif
(!!args[\"critical_threshold\"] && percentOK <=
args[\"critical_threshold\"]) {\n    event.check.output
```

```
= \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of
selected events are OK (\" + percentOK.toString() +
\"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] &&
percentOK <= args[\"warning_threshold\"]) {\n
event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of
selected events are OK (\" + percentOK.toString() +
\"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n
crit = sensu.CountBySeverity(\"critical\");\n\n    if
(crit >= args[\"critical_count\"]) {\n
event.check.output = \"CRITICAL: \" +
args[\"critical_count\"].toString() + \" or more
selected events are in a critical state (\" +
crit.toString() + \")\n\";\n        event.check.status
= 2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn =
sensu.CountBySeverity(\"warning\");\n\n    if (warn >=
args[\"warning_count\"]) {\n        event.check.output
= \"WARNING: \" + args[\"warning_count\"].toString() +
\" or more selected events are in a warning state (\" +
warn.toString() + \")\n\";\n        event.check.status
= 1;\n        return event;\n
}\n}\n\nevent.check.output = \"Everything looks good
(\" + percentOK.toString() + \"%
OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific rule template

The `/rule-templates/:rule-template` API endpoint provides HTTP GET access to data for a specific rule template by name.

Example

The following example queries the `/rule-templates/:rule-template` API endpoint for a specific `:rule-template`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/rule-
templates/aggregate \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:rule-template` definition (in this example, `aggregate`):

```
{
  "type": "RuleTemplate",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "aggregate",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a critical status if the percentage
of non-zero events is equal to or greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
```

```
          "default": {},
          "description": "metric handlers to use for produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from aggregate data and include them in
the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a warning status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event with a warning status if the percentage of
non-zero events is equal to or greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service - aggregate one or more events
into a single event. This BSM rule template allows you to treat the results of
multiple disparate check executions – executed across multiple disparate systems –
as a single event. This template is extremely useful in dynamic environments and/or
environments that have a reasonable tolerance for failure. Use this template when a
service can be considered healthy as long as a minimum threshold is satisfied (for
example, at least 5 healthy web servers? at least 70% of N processes healthy?).",
    "eval": "\nif (events && events.length == 0) {\n    event.check.output =
\"WARNING: No events selected for aggregate\n\";\n    event.check.status = 1;\n
return event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_event_count\"] =
events.length;\n\npercentOK = sensu.PercentageBySeverity(\"ok\");\n\nif
```

```
(!!args[\"produce_metrics\"]) {\n    var handlers = [];\n\n    if
(!!args[\"metric_handlers\"]) {\n        handlers =
args[\"metric_handlers\"].slice();\n    }\n\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n    event.timestamp = ts;\n\n    var tags = [\n
{\n            name: \"service\",\n            value: event.entity.name\n
},\n        {\n            name: \"entity\",\n            value: event.entity.name\n
},\n        {\n            name: \"check\",\n            value: event.check.name\n
}\n    ];\n\n    event.metrics = sensu.NewMetrics({\n        handlers: handlers,\n
points: [\n            {\n                name: \"percent_non_zero\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n                name: \"percent_ok\",\n
timestamp: ts,\n                value: percentOK,\n                tags: tags\n
},\n            {\n                name: \"percent_warning\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n                name:
\"percent_critical\",\n                timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n                tags: tags\n
},\n            {\n                name: \"percent_unknown\",\n
timestamp: ts,\n                value: sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n                name:
\"count_non_zero\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"non-zero\"),\n                tags: tags\n            },\n
{\n                name: \"count_ok\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"ok\"),\n                tags: tags\n            },\n
{\n                name: \"count_warning\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"warning\"),\n                tags: tags\n
},\n            {\n                name: \"count_critical\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n                name:
\"count_unknown\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n                tags: tags\n            }\n
]\n    });\n\n    if (!!args[\"set_metric_annotations\"]) {\n        var i = 0;\n\n
while(i < event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n            i++;\n        }\n
}\n}\n\nif (!!args[\"critical_threshold\"] && percentOK <=
args[\"critical_threshold\"]) {\n    event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] && percentOK <=
args[\"warning_threshold\"]) {\n    event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected events are OK (\" +
```

```
percentOK.toString() + \"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit >= args[\"critical_count\"])
{\n        event.check.output = \"CRITICAL: \" + args[\"critical_count\"].toString()
+ \" or more selected events are in a critical state (\" + crit.toString() +
\")\n\";\n        event.check.status = 2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn = sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n        event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected events are in a warning
state (\" + warn.toString() + \")\n\";\n        event.check.status = 1;\n
return event;\n    }\n}\n\nevent.check.output = \"Everything looks good (\" +
percentOK.toString() + \"% OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}
```

API Specification

/rule-templates/:rule-template (GET)	
description	Returns the specified rule template.
example url	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates/aggregate
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "RuleTemplate", "api_version": "bsm/v1", "metadata": { "name": "aggregate", "namespace": "default", "created_by": "admin"

```
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event
with a critical status if there the number
of critical events is equal to or greater
than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event
with a critical status if the percentage of
non-zero events is equal to or greater than
this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers
to use for produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics
from aggregate data and include them in the
produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the
produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event
```

```
with a warning status if there the number
of critical events is equal to or greater
than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event
with a warning status if the percentage of
non-zero events is equal to or greater than
this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed
service - aggregate one or more events into
a single event. This BSM rule template
allows you to treat the results of multiple
disparate check executions – executed
across multiple disparate systems – as a
single event. This template is extremely
useful in dynamic environments and/or
environments that have a reasonable
tolerance for failure. Use this template
when a service can be considered healthy as
long as a minimum threshold is satisfied
(for example, at least 5 healthy web
servers? at least 70% of N processes
healthy?).",
    "eval": "\nif (events && events.length
== 0) {\n    event.check.output =
\"WARNING: No events selected for
aggregate\n\";\n    event.check.status =
1;\n    return
event;\n}\n\nevent.annotations[\"io.sensu.b
sm.selected_event_count\"] =
events.length;\n\npercentOK =
sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var
handlers = [];\n\n    if
(!!args[\"metric_handlers\"]) {\n
```

```
handlers =
args[\"metric_handlers\"].slice();\n
}\n\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n
event.timestamp = ts;\n\n    var tags = [\n
{\n            name: \"service\",\n
value: event.entity.name\n        },\n
{\n            name: \"entity\",\n
value: event.entity.name\n        },\n
{\n            name: \"check\",\n
value: event.check.name\n        }\n
];\n\n    event.metrics =
sensu.NewMetrics({\n        handlers:
handlers,\n        points: [\n
{\n                name:
\"percent_non_zero\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"percent_ok\",\n
timestamp: ts,\n                value:
percentOK,\n                tags: tags\n
},\n            {\n                name:
\"percent_warning\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"percent_critical\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"critical\"),\n
                tags: tags\n
},\n            {\n                name:
\"percent_unknown\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n            },\n            {\n
name: \"count_non_zero\",\n
timestamp: ts,\n                value:
sensu.CountBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"count_ok\",\n
timestamp: ts,\n                value:
```

```
sensu.CountBySeverity(\"ok\"),\n
tags: tags\n            },\n            {\n
name: \"count_warning\",\n
timestamp: ts,\n                value:
sensu.CountBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"count_critical\",\n
timestamp: ts,\n                value:
sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n
name: \"count_unknown\",\n
timestamp: ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n            }\n        ]\n
});\n\n    if
(!!args[\"set_metric_annotations\"]) {\n
var i = 0;\n\n        while(i <
event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_e
vent_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n
            i++;\n        }\n    }\n}\n\nif
(!!args[\"critical_threshold\"] &&
percentOK <= args[\"critical_threshold\"])
{\n    event.check.output = \"CRITICAL:
Less than \" +
args[\"critical_threshold\"].toString() +
\"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n
event.check.status = 2;\n    return
event;\n}\n\nif
(!!args[\"warning_threshold\"] && percentOK
<= args[\"warning_threshold\"]) {\n
event.check.output = \"WARNING: Less than
\" + args[\"warning_threshold\"].toString()
+ \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n
event.check.status = 1;\n    return
event;\n}\n\nif
(!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n
if (crit >= args[\"critical_count\"]) {\n
```

```
event.check.output = \"CRITICAL: \" +
args[\"critical_count\"].toString() + \" or
more selected events are in a critical
state (\" + crit.toString() + \")\n\";\n
event.check.status = 2;\n        return
event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn =
sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n
event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or
more selected events are in a warning state
(\" + warn.toString() + \")\n\";\n
event.check.status = 1;\n        return
event;\n    }\n}\n\nevent.check.output =
\"Everything looks good (\" +
percentOK.toString() + \"%
OK)\";\nevent.check.status = 0;\n\nreturn
event;\n"
  }
}
```

Create or update a rule template

The `/rule-templates/:rule-template` API endpoint provides HTTP PUT access to create or update a specific rule template by name.

Example

The following example demonstrates a request to the `/rule-templates/:rule-template` API endpoint to update the rule template `aggregate`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "RuleTemplate",
```

```
  "api_version": "bsm/v1",
  "metadata": {
    "name": "aggregate"
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a critical status if the percentage
of non-zero events is equal to or greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers to use for produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
          "description": "produce metrics from aggregate data and include them in
the produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event with a warning status if there the number
of critical events is equal to or greater than this count",
          "type": "number"
        },
        "warning_threshold": {
```

```
          "description": "create an event with a warning status if the percentage of non-zero events is equal to or greater than this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed service - aggregate one or more events into a single event. This BSM rule template allows you to treat the results of multiple disparate check executions - executed across multiple disparate systems - as a single event. This template is extremely useful in dynamic environments and/or environments that have a reasonable tolerance for failure. Use this template when a service can be considered healthy as long as a minimum threshold is satisfied (for example, at least 5 healthy web servers? at least 70% of N processes healthy?).",
    "eval": "\nif (events && events.length == 0) {\n    event.check.output = \"WARNING: No events selected for aggregate\n\";\n    event.check.status = 1;\n return event;\n}\n\nevent.annotations[\"io.sensu.bsm.selected_event_count\"] = events.length;\n\npercentOK = sensu.PercentageBySeverity(\"ok\");\n\nif (!!args[\"produce_metrics\"]) {\n    var handlers = [];\n\n    if (!!args[\"metric_handlers\"]) {\n        handlers = args[\"metric_handlers\"].slice();\n    }\n\n    var ts = Math.floor(new Date().getTime() / 1000);\n\n    event.timestamp = ts;\n\n    var tags = [\n {\n            name: \"service\",\n            value: event.entity.name\n },\n        {\n            name: \"entity\",\n            value: event.entity.name\n },\n        {\n            name: \"check\",\n            value: event.check.name\n }\n    ];\n\n    event.metrics = sensu.NewMetrics({\n        handlers: handlers,\n points: [\n            {\n                name: \"percent_non_zero\",\n timestamp: ts,\n                value: sensu.PercentageBySeverity(\"non-zero\"),\n tags: tags\n            },\n            {\n                name: \"percent_ok\",\n timestamp: ts,\n                value: percentOK,\n                tags: tags\n },\n            {\n                name: \"percent_warning\",\n timestamp: ts,\n                value: sensu.PercentageBySeverity(\"warning\"),\n tags: tags\n            },\n            {\n                name: \"percent_critical\",\n                timestamp: ts,\n                value: sensu.PercentageBySeverity(\"critical\"),\n                tags: tags\n },\n            {\n                name: \"percent_unknown\",\n timestamp: ts,\n                value: sensu.PercentageBySeverity(\"unknown\"),\n tags: tags\n            },\n            {\n                name: \"count_non_zero\",\n                timestamp: ts,\n                value: sensu.CountBySeverity(\"non-zero\"),\n                tags: tags\n            },\n {\n                name: \"count_ok\",\n                timestamp: ts,\n value: sensu.CountBySeverity(\"ok\"),\n                tags: tags\n            },\n
```

```
{\n                name: \"count_warning\",\n                timestamp: ts,\n
value: sensu.CountBySeverity(\"warning\"),\n                tags: tags\n
},\n            {\n                name: \"count_critical\",\n
timestamp: ts,\n                value: sensu.CountBySeverity(\"critical\"),\n
tags: tags\n            },\n            {\n                name:
\"count_unknown\",\n                timestamp: ts,\n                value:
sensu.CountBySeverity(\"unknown\"),\n                tags: tags\n            }\n
]\n    });\n\n    if (!!args[\"set_metric_annotations\"]) {\n        var i = 0;\n\n
while(i < event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_event_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n            i++;\n        }\n
}\n}\n\nif (!!args[\"critical_threshold\"] && percentOK <=
args[\"critical_threshold\"]) {\n    event.check.output = \"CRITICAL: Less than \" +
args[\"critical_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 2;\n    return
event;\n}\n\nif (!!args[\"warning_threshold\"] && percentOK <=
args[\"warning_threshold\"]) {\n    event.check.output = \"WARNING: Less than \" +
args[\"warning_threshold\"].toString() + \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n    event.check.status = 1;\n    return
event;\n}\n\nif (!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n    if (crit >= args[\"critical_count\"])
{\n        event.check.output = \"CRITICAL: \" + args[\"critical_count\"].toString()
+ \" or more selected events are in a critical state (\" + crit.toString() +
\")\n\";\n        event.check.status = 2;\n        return event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn = sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n        event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or more selected events are in a warning
state (\" + warn.toString() + \")\n\";\n        event.check.status = 1;\n
return event;\n    }\n}\n\nevent.check.output = \"Everything looks good (\" +
percentOK.toString() + \"% OK)\";\nevent.check.status = 0;\n\nreturn event;\n"
  }
}' \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/rule-
templates/aggregate
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/rule-templates/:rule-template (PUT)	
description	Creates or updates the specified rule template.
example URL	http://hostname:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates/aggregate
payload	


```
{
  "type": "RuleTemplate",
  "api_version": "bsm/v1",
  "metadata": {
    "name": "aggregate"
  },
  "spec": {
    "arguments": {
      "properties": {
        "critical_count": {
          "description": "create an event with a critical status if there the number of critical events is equal to or greater than this count",
          "type": "number"
        },
        "critical_threshold": {
          "description": "create an event with a critical status if the percentage of non-zero events is equal to or greater than this threshold",
          "type": "number"
        },
        "metric_handlers": {
          "default": {},
          "description": "metric handlers to use for produced metrics",
          "items": {
            "type": "string"
          },
          "type": "array"
        },
        "produce_metrics": {
          "default": {},
```

```
          "description": "produce metrics
from aggregate data and include them in the
produced event",
          "type": "boolean"
        },
        "set_metric_annotations": {
          "default": {},
          "description": "annotate the
produced event with metric annotations",
          "type": "boolean"
        },
        "warning_count": {
          "description": "create an event
with a warning status if there the number
of critical events is equal to or greater
than this count",
          "type": "number"
        },
        "warning_threshold": {
          "description": "create an event
with a warning status if the percentage of
non-zero events is equal to or greater than
this threshold",
          "type": "number"
        }
      },
      "required": null
    },
    "description": "Monitor a distributed
service - aggregate one or more events into
a single event. This BSM rule template
allows you to treat the results of multiple
disparate check executions - executed
across multiple disparate systems - as a
single event. This template is extremely
useful in dynamic environments and/or
environments that have a reasonable
tolerance for failure. Use this template
when a service can be considered healthy as
long as a minimum threshold is satisfied
(for example, at least 5 healthy web
servers? at least 70% of N processes
```

```
healthy?).",
    "eval": "\nif (events && events.length
== 0) {\n    event.check.output =
\"WARNING: No events selected for
aggregate\n\";\n    event.check.status =
1;\n    return
event;\n}\n\nevent.annotations[\"io.sensu.b
sm.selected_event_count\"] =
events.length;\n\npercentOK =
sensu.PercentageBySeverity(\"ok\");\n\nif
(!!args[\"produce_metrics\"]) {\n    var
handlers = [];\n\n    if
(!!args[\"metric_handlers\"]) {\n
handlers =
args[\"metric_handlers\"].slice();\n
}\n\n    var ts = Math.floor(new
Date().getTime() / 1000);\n\n
event.timestamp = ts;\n\n    var tags = [\n
{\n            name: \"service\",\n
value: event.entity.name\n        },\n
{\n            name: \"entity\",\n
value: event.entity.name\n        },\n
{\n            name: \"check\",\n
value: event.check.name\n        }\n
];\n\n    event.metrics =
sensu.NewMetrics({\n        handlers:
handlers,\n        points: [\n
{\n                name:
\"percent_non_zero\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"non-zero\"),\n
tags: tags\n            },\n            {\n
name: \"percent_ok\",\n
timestamp: ts,\n                value:
percentOK,\n                tags: tags\n
},\n            {\n                name:
\"percent_warning\",\n
timestamp: ts,\n                value:
sensu.PercentageBySeverity(\"warning\"),\n
tags: tags\n            },\n            {\n
name: \"percent_critical\",\n
timestamp: ts,\n                value:
```

```
sensu.PercentageBySeverity(\"critical\"),\n
                tags: tags\n
},\n             {\n                 name:
\"percent_unknown\",\n
timestamp: ts,\n                 value:
sensu.PercentageBySeverity(\"unknown\"),\n
tags: tags\n             },\n             {\n
name: \"count_non_zero\",\n
timestamp: ts,\n                 value:
sensu.CountBySeverity(\"non-zero\"),\n
tags: tags\n             },\n             {\n
name: \"count_ok\",\n
timestamp: ts,\n                 value:
sensu.CountBySeverity(\"ok\"),\n
tags: tags\n             },\n             {\n
name: \"count_warning\",\n
timestamp: ts,\n                 value:
sensu.CountBySeverity(\"warning\"),\n
tags: tags\n             },\n             {\n
name: \"count_critical\",\n
timestamp: ts,\n                 value:
sensu.CountBySeverity(\"critical\"),\n
tags: tags\n             },\n             {\n
name: \"count_unknown\",\n
timestamp: ts,\n                 value:
sensu.CountBySeverity(\"unknown\"),\n
tags: tags\n             }\n         ]\n
});\n\n    if
(!!args[\"set_metric_annotations\"]) {\n
var i = 0;\n\n         while(i <
event.metrics.points.length) {\n
event.annotations[\"io.sensu.bsm.selected_e
vent_\" + event.metrics.points[i].name] =
event.metrics.points[i].value.toString();\n
            i++;\n         }\n    }\n}\n\nif
(!!args[\"critical_threshold\"] &&
percentOK <= args[\"critical_threshold\"])
{\n    event.check.output = \"CRITICAL:
Less than \" +
args[\"critical_threshold\"].toString() +
\"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n
```

```
event.check.status = 2;\n    return
event;\n}\n\nif
(!!args[\"warning_threshold\"] && percentOK
<= args[\"warning_threshold\"]) {\n
event.check.output = \"WARNING: Less than
\" + args[\"warning_threshold\"].toString()
+ \"% of selected events are OK (\" +
percentOK.toString() + \"%)\n\";\n
event.check.status = 1;\n    return
event;\n}\n\nif
(!!args[\"critical_count\"]) {\n    crit =
sensu.CountBySeverity(\"critical\");\n\n
if (crit >= args[\"critical_count\"]) {\n
event.check.output = \"CRITICAL: \" +
args[\"critical_count\"].toString() + \" or
more selected events are in a critical
state (\" + crit.toString() + \")\n\";\n
event.check.status = 2;\n        return
event;\n    }\n}\n\nif
(!!args[\"warning_count\"]) {\n    warn =
sensu.CountBySeverity(\"warning\");\n\n
if (warn >= args[\"warning_count\"]) {\n
event.check.output = \"WARNING: \" +
args[\"warning_count\"].toString() + \" or
more selected events are in a warning state
(\" + warn.toString() + \")\n\";\n
event.check.status = 1;\n        return
event;\n    }\n}\n\nevent.check.output =
\"Everything looks good (\" +
percentOK.toString() + \"%
OK)\";\nevent.check.status = 0;\n\nreturn
event;\n"
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a rule template

The `/rule-templates/:rule-template` API endpoint provides HTTP DELETE access to delete the specified rule template from Sensu.

Example

The following example shows a request to the `/rule-templates/:rule-template` API endpoint to delete the rule template `aggregate`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/bsm/v1/namespaces/default/rule-templates/aggregate
```

API Specification

/rule-templates/:rule-template (DELETE)	
description	Deletes the specified rule template from Sensu.
example url	http://hostname:8080/api/enterprise/bsm/v1/rule-templates/aggregate
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

enterprise/federation/v1

***COMMERCIAL FEATURE**: Access federation in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/federation/v1` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all replicators

The `/etcd-replicators` API endpoint provides HTTP GET access to a list of replicators.

***NOTE**: The etcd-replicators datatype is only accessible for users who have a cluster role that permits access to replication resources.*

Example

The following example demonstrates a GET request to the `/etcd-replicators` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the etcd replicator definitions:

```
[
  {
```

```
    "api_version": "federation/v1",
    "type": "EtcdReplicator",
    "metadata": {
      "name": "my_replicator",
      "created_by": "admin"
    },
    "spec": {
      "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
      "cert": "/path/to/ssl/cert.pem",
      "key": "/path/to/ssl/key.pem",
      "insecure": false,
      "url": "http://remote-etcd.example.com:2379",
      "api_version": "core/v2",
      "resource": "Role",
      "replication_interval_seconds": 30
    }
  }
]
```

API Specification

/etcd-replicators (GET)	
description	Returns the list of replicators.
example url	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "api_version": "federation/v1", "type": "EtcdReplicator", "metadata": {

```
      "name": "my_replicator",
      "created_by": "admin"
    },
    "spec": {
      "ca_cert": "/path/to/ssl/trusted-certificate-
authorities.pem",
      "cert": "/path/to/ssl/cert.pem",
      "key": "/path/to/ssl/key.pem",
      "insecure": false,
      "url": "http://remote-etcd.example.com:2379",
      "api_version": "core/v2",
      "resource": "Role",
      "replication_interval_seconds": 30
    }
  }
]
```

Create a new replicator

The `/etcd-replicators` API endpoint provides HTTP POST access to create replicators.

> ***NOTE**: Create a replicator for each resource type you want to replicate.Replicating `namespace` resources will **not** replicate the resources that belong to those namespaces.*

Example

The following example demonstrates a request to the `/etcd-replicators` API endpoint to create the replicator `my_replicator`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
```

```
  "metadata": {
    "name": "my_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://remote-etcd.example.com:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}' \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/etcd-replicators (POST)	
description	Creates a new replicator (if none exists).
example URL	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators
payload	{ "api_version": "federation/v1", "type": "EtcdReplicator", "metadata": { "name": "my_replicator" }, "spec": { "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem", "cert": "/path/to/ssl/cert.pem", "key": "/path/to/ssl/key.pem", "insecure": false, "url": "http://remote-etcd.example.com:2379",

```
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific replicator

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP GET access to data for a specific `:etcd-replicator`, by replicator name.

> ***NOTE**: The etcd-replicators datatype is only accessible for users who have a cluster role that permits access to replication resources.*

Example

The following example queries the `/etcd-replicators/:etcd-replicator` API endpoint for a specific `:etcd-replicator`.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:etcd-replicator` definition (in this example, `my_replicator`):

```
{
```

```
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "my_replicator",
    "created_by": "admin"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://remote-etcd.example.com:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

API Specification

/etcd-replicators/:etcd-replicator (GET)	
description	Returns the specified replicator.
example url	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "api_version": "federation/v1", "type": "EtcdReplicator", "metadata": {

```
    "name": "my_replicator",
    "created_by": "admin"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-
certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://remote-
etcd.example.com:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

Create or update a replicator

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP PUT access to create or update a specific `:etcd-replicator`, by replicator name.

Example

The following example demonstrates a request to the `/etcd-replicators/:etcd-replicator` API endpoint to update the replicator `my_replicator`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "my_replicator"
  },
  "spec": {
```

```
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://remote-etcd.example.com:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}' \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my-replicator
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/etcd-replicators/:etcd-replicator (PUT)	
description	Creates or updates the specified replicator. The replicator resource and API version cannot be altered.
example URL	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator
payload	{ "api_version": "federation/v1", "type": "EtcdReplicator", "metadata": { "name": "my_replicator" }, "spec": { "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem", "cert": "/path/to/ssl/cert.pem", "key": "/path/to/ssl/key.pem", "insecure": **false**, "url": "http://remote-etcd.example.com:2379",

```
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a replicator

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP DELETE access to delete the specified replicator from Sensu.

Example

The following example shows a request to the `/etcd-replicators/:etcd-replicator` API endpoint to delete the replicator `my_replicator`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator
```

API Specification

/etcd-replicators/:etcd-replicator (DELETE)	
description	Deletes the specified replicator from Sensu.
example url	http://hostname:8080/api/enterprise/federation

response codes

- **Success**: 204 (No Content)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get all clusters

The `/clusters` API endpoint provides HTTP GET access to a list of clusters.

Example

The following example demonstrates a request to the `/clusters` API endpoint, resulting in a list of clusters.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the cluster definitions:

```
[
  {
    "type": "Cluster",
    "api_version": "federation/v1",
    "metadata": {
      "name": "us-west-2a",
      "created_by": "admin"
    },
    "spec": {
      "api_urls": [
        "http://10.0.0.1:8080",
        "http://10.0.0.2:8080",
```

```
          "http://10.0.0.3:8080"
        ]
      }
    }
]
```

API Specification

/clusters (GET)	
description	Returns the list of clusters.
example url	http://hostname:8080/api/enterprise/federation/v1/clusters
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "type": "Cluster", "api_version": "federation/v1", "metadata": { "name": "us-west-2a", "created_by": "admin" }, "spec": { "api_urls": ["http://10.0.0.1:8080", "http://10.0.0.2:8080", "http://10.0.0.3:8080"] } }]

Get a specific cluster

The `/clusters/:cluster` API endpoint provides HTTP GET access to data for a specific `cluster`, by cluster name.

Example

The following example queries the `/clusters/:cluster` API endpoint for a specific `:cluster`.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:cluster` definition (in this example, `us-west-2a`):

```
{
  "type": "Cluster",
  "api_version": "federation/v1",
  "metadata": {
    "name": "us-west-2a",
    "created_by": "admin"
  },
  "spec": {
    "api_urls": [
      "http://10.0.0.1:8080",
      "http://10.0.0.2:8080",
      "http://10.0.0.3:8080"
    ]
  }
}
```

API Specification

/clusters/:cluster (GET)	
description	Returns the specified cluster.
example url	http://hostname:8080/api/enterprise/federation/v1/clusters/us-west-2a
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "Cluster", "api_version": "federation/v1", "metadata": { "name": "us-west-2a", "created_by": "admin" }, "spec": { "api_urls": ["http://10.0.0.1:8080", "http://10.0.0.2:8080", "http://10.0.0.3:8080"] } }

Create or update a cluster

The `/clusters/:cluster` API endpoint provides HTTP PUT access to create or update a specific `cluster`, by cluster name.

***NOTE**: Only cluster admins have PUT access to clusters.*

Example

The following example demonstrates a request to the `/clusters/:cluster` API endpoint to update the cluster `us-west-2a`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "Cluster",
  "api_version": "federation/v1",
  "metadata": {
    "name": "us-west-2a"
  },
  "spec": {
    "api_urls": [
      "http://10.0.0.1:8080",
      "http://10.0.0.2:8080",
      "http://10.0.0.3:8080"
    ]
  }
}' \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/clusters/:cluster (PUT)	
description	Creates or updates the specified cluster.
example URL	http://hostname:8080/api/enterprise/federation/v1/clusters/us-west-2a

payload

```
{
  "type": "Cluster",
  "api_version": "federation/v1",
  "metadata": {
    "name": "us-west-2a"
  },
  "spec": {
    "api_urls": [
      "http://10.0.0.1:8080",
      "http://10.0.0.2:8080",
      "http://10.0.0.3:8080"
    ]
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a cluster

The `/clusters/:cluster` API endpoint provides HTTP DELETE access to delete the specified cluster from Sensu.

> ***NOTE**: Only cluster admins have DELETE access to clusters.*

Example

The following example shows a request to the `/clusters/:cluster` API endpoint to delete the cluster `us-west-2a`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
```

```
  -H "Authorization: Key $SENSU_API_KEY" \
  http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a
```

API Specification

/clusters/:cluster (DELETE)	
description	Deletes the specified cluster from Sensu.
example url	http://hostname:8080/api/enterprise/federation/v1/clusters/us-west-2a
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

enterprise/pipeline/v1

***IMPORTANT**: The `enterprise/pipeline/v1` API endpoints do not allow you to create and manage pipelines, which are composed of observation event processing workflows. Instead, `enterprise/pipeline/v1` API endpoints allow you to create and manage resources that can **only** be used within pipelines (the Sumo Logic metrics handlers and TCP stream handlers).*

***COMMERCIAL FEATURE**: Access `enterprise/pipeline/v1` API endpoints in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/pipeline/v1` endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all Sumo Logic metrics handler resources

The `/sumo-logic-metrics-handlers` API endpoint provides HTTP GET access to Sumo Logic metrics handler data.

Example

The following example demonstrates a GET request to the `/sumo-logic-metrics-handlers` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the Sumo Logic metrics handler definitions in the `default` namespace:

```
[
  {
    "type": "SumoLogicMetricsHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name": "sumologic_http_log_metrics_us1",
      "namespace": "default"
    },
    "spec": {
      "url": "$SUMO_LOGIC_SOURCE_URL",
      "secrets": [
        {
          "name": "SUMO_LOGIC_SOURCE_URL",
          "secret": "sumologic_metrics_us1"
        }
      ],
      "max_connections": 10,
      "timeout": "30s"
    }
  },
  {
    "type": "SumoLogicMetricsHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name": "sumologic_http_log_metrics_us2",
      "namespace": "default"
    },
    "spec": {
      "url": "$SUMO_LOGIC_SOURCE_URL",
      "secrets": [
        {
          "name": "SUMO_LOGIC_SOURCE_URL",
          "secret": "sumologic_metrics_us2"
        }
      ],
      "max_connections": 10,
      "timeout": "30s"
    }
  }
]
```

API Specification

/sumo-logic-metrics-handlers (GET)	
description	Returns the list of Sumo Logic metrics handlers.
example url	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "type": "SumoLogicMetricsHandler", "api_version": "pipeline/v1", "metadata": { "name": "sumologic_http_log_metrics_us1", "namespace": "default" }, "spec": { "url": "$SUMO_LOGIC_SOURCE_URL", "secrets": [{ "name": "SUMO_LOGIC_SOURCE_URL", "secret": "sumologic_metrics_us1" }], "max_connections": 10,

```
      "timeout": "30s"
    }
  },
  {
    "type": "SumoLogicMetricsHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name":
"sumologic_http_log_metrics_us2",
      "namespace": "default"
    },
    "spec": {
      "url": "$SUMO_LOGIC_SOURCE_URL",
      "secrets": [
        {
          "name": "SUMO_LOGIC_SOURCE_URL",
          "secret": "sumologic_metrics_us2"
        }
      ],
      "max_connections": 10,
      "timeout": "30s"
    }
  }
]
```

Create a new Sumo Logic metrics handler

The `/sumo-logic-metrics-handlers` API endpoint provides HTTP POST access to create a Sumo Logic metrics handler.

Example

In the following example, an HTTP POST request is submitted to the `/sumo-logic-metrics-handlers` API endpoint to create the Sumo Logic metrics handler `sumologic_http_log_metrics_us1`:

```
curl -X POST \
```

```
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumologic_http_log_metrics_us1"
  },
  "spec": {
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
        "secret": "sumologic_metrics_us1"
      }
    ],
    "max_connections": 10,
    "timeout": "30s"
  }
}' \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/sumo-logic-metrics-handlers (POST)	
description	Creates a Sensu Sumo Logic metrics handler.
example URL	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers
payload	{ "type": "SumoLogicMetricsHandler", "api_version": "pipeline/v1", "metadata": { "name":

```
"sumologic_http_log_metrics_us1"
  },
  "spec": {
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
        "secret": "sumologic_metrics_us1"
      }
    ],
    "max_connections": 10,
    "timeout": "30s"
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific Sumo Logic metrics handler

The `/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler` API endpoint provides HTTP GET access to Sumo Logic metrics handler data for specific `:sumo-logic-metrics-handler` definitions, by handler `name`.

Example

The following example queries the `/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler` API endpoint for the `:sumo-logic-metrics-handler` named `sumologic_http_log_metrics_us1`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers/sumologic_http_log_metrics_us1 \
```

```
  -H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:sumo-logic-metrics-handler` definition (in this example, `sumologic_http_log_metrics_us1`):

```
{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumologic_http_log_metrics_us1",
    "namespace": "default"
  },
  "spec": {
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
        "secret": "sumologic_metrics_us1"
      }
    ],
    "max_connections": 10,
    "timeout": "30s"
  }
}
```

API Specification

/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler (GET)	
description	Returns a Sumo Logic metrics handler.
example url	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-

	handlers/sumologic_http_log_metrics_us1
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "SumoLogicMetricsHandler", "api_version": "pipeline/v1", "metadata": { "name": "sumologic_http_log_metrics_us1", "namespace": "default" }, "spec": { "url": "$SUMO_LOGIC_SOURCE_URL", "secrets": [{ "name": "SUMO_LOGIC_SOURCE_URL", "secret": "sumologic_metri

```
cs_us1"
      }
    ],

"max_connections
": 10,
    "timeout":
"30s"
  }
}
```

Create or update a Sumo Logic metrics handler

The `/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler` API endpoint provides HTTP PUT access to create or update a specific `:sumo-logic-metrics-handler` definition, by handler name.

Example

In the following example, an HTTP PUT request is submitted to the `/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler` API endpoint to create `sumologic_http_log_metrics_us2`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "SumoLogicMetricsHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "sumologic_http_log_metrics_us2"
  },
  "spec": {
    "url": "$SUMO_LOGIC_SOURCE_URL",
    "secrets": [
      {
        "name": "SUMO_LOGIC_SOURCE_URL",
```

```
        "secret": "sumologic_metrics_us2"
      }
    ],
    "max_connections": 10,
    "timeout": "30s"
  }
}' \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers/sumologic_http_log_metrics_us2
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler (PUT)	
description	Creates or updates the specified Sensu Sumo Logic metrics handler.
example URL	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers/sumologic_http_log_metrics_us2
payload	{ "type": "SumoLogicMetricsHandler", "api_version": "pipeline/v1", "metadata": { "name": "sumologic_http_log_metrics_us2"

```
  },
  "spec": {
    "url":
"$SUMO_LOGIC_SOU
RCE_URL",
    "secrets": [
      {
        "name":
"SUMO_LOGIC_SOUR
CE_URL",

"secret":
"sumologic_metri
cs_us2"
      }
    ],

"max_connections
": 10,
    "timeout":
"30s"
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a Sumo Logic metrics handler

The `/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler` API endpoint provides HTTP

DELETE access to delete a Sumo Logic metrics handler from Sensu (specified by the handler name).

Example

The following example shows a request to the `/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler` API endpoint to delete the Sumo Logic metrics handler `sumologic_http_log_metrics_us2`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers/sumologic_http_log_metrics_us2 \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/sumo-logic-metrics-handlers/:sumo-logic-metrics-handler (DELETE)	
description	Removes the specified Sumo Logic metrics handler from Sensu.
example url	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/sumo-logic-metrics-handlers/sumologic_http_log_metrics_us2
response codes	¬ **Success**: 204 (No Content)

- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get all TCP stream handler resources

The `/tcp-stream-handlers` API endpoint provides HTTP GET access to TCP stream handler data.

Example

The following example demonstrates a GET request to the `/tcp-stream-handlers` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the TCP stream handler definitions in the `default` namespace:

```
[
  {
    "type": "TCPStreamHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name": "incident_log",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
```

```
      "address": "127.0.0.1:4242",
      "max_connections": 10,
      "max_reconnect_delay": "10s",
      "min_reconnect_delay": "10ms",
      "tls_ca_cert_file": "",
      "tls_cert_file": "",
      "tls_key_file": ""
    }
  },
  {
    "type": "TCPStreamHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name": "logstash",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "address": "127.0.0.1:4242",
      "max_connections": 10,
      "max_reconnect_delay": "10s",
      "min_reconnect_delay": "10ms",
      "tls_ca_cert_file": "/path/to/tls/ca.pem",
      "tls_cert_file": "/path/to/tls/cert.pem",
      "tls_key_file": "/path/to/tls/key.pem"
    }
  }
]
```


API Specification

/tcp-stream-handlers (GET)	
description	Returns the list of TCP stream handlers.
example url	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.

response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "type": "TCPStreamHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name": "incident_log",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "address": "127.0.0.1:4242",
      "max_connections": 10,
      "max_reconnect_delay": "10s",
      "min_reconnect_delay": "10ms",
      "tls_ca_cert_file": "",
      "tls_cert_file": "",
      "tls_key_file": ""
    }
  },
  {
    "type": "TCPStreamHandler",
    "api_version": "pipeline/v1",
    "metadata": {
      "name": "logstash",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "address": "127.0.0.1:4242",
      "max_connections": 10,
      "max_reconnect_delay": "10s",
      "min_reconnect_delay": "10ms",
```

```
      "tls_ca_cert_file": "/path/to/tls/ca.pem",
      "tls_cert_file": "/path/to/tls/cert.pem",
      "tls_key_file": "/path/to/tls/key.pem"
    }
  }
]
```

Create a new TCP stream handler

The `/tcp-stream-handlers` API endpoint provides HTTP POST access to create a TCP stream handler.

Example

In the following example, an HTTP POST request is submitted to the `/tcp-stream-handlers` API endpoint to create the TCP stream handler `logstash`:

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "api_version": "pipeline/v1",
  "type": "TCPStreamHandler",
  "metadata": {
    "name": "logstash"
  },
  "spec": {
    "address": "127.0.0.1:4242",
    "tls_ca_cert_file": "/path/to/tls/ca.pem",
    "tls_cert_file": "/path/to/tls/cert.pem",
    "tls_key_file": "/path/to/tls/key.pem",
    "max_connections": 10,
    "min_reconnect_delay": "10ms",
    "max_reconnect_delay": "10s"
  }
}' \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-
```

```
handlers
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/tcp-stream-handlers (POST)	
description	Creates a Sensu TCP stream handler.
example URL	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers
payload	{ "api_version": "pipeline/v1", "type": "TCPStreamHandler", "metadata": { "name": "logstash" }, "spec": { "address": "127.0.0.1:4242", "tls_ca_cert_file": "/path/to/tls/ca.pem", "tls_cert_file": "/path/to/tls/cert.pem", "tls_key_file": "/path/to/tls/key.pem", "max_connections": 10, "min_reconnect_delay": "10ms", "max_reconnect_delay": "10s" } }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific TCP stream handler

The `/tcp-stream-handlers/:tcp-stream-handler` API endpoint provides HTTP GET access to TCP stream handler data for specific `:tcp-stream-handler` definitions, by handler `name`.

Example

The following example queries the `/tcp-stream-handlers/:tcp-stream-handler` API endpoint for the `:tcp-stream-handler` named `logstash`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers/logstash \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:tcp-stream-handler` definition (in this example, `logstash`):

```
{
  "type": "TCPStreamHandler",
  "api_version": "pipeline/v1",
  "metadata": {
    "name": "logstash",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "address": "127.0.0.1:4242",
    "max_connections": 10,
    "max_reconnect_delay": "10s",
    "min_reconnect_delay": "10ms",
    "tls_ca_cert_file": "/path/to/tls/ca.pem",
    "tls_cert_file": "/path/to/tls/cert.pem",
    "tls_key_file": "/path/to/tls/key.pem"
  }
}
```

API Specification

/tcp-stream-handlers/:tcp-stream-handler (GET)	
description	Returns a TCP stream handler.
example url	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers/logstash
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "TCPStreamHandler", "api_version": "pipeline/v1", "metadata": { "name": "logstash", "namespace": "default", "created_by": "admin" }, "spec": { "address": "127.0.0.1:4242", "max_connections": 10, "max_reconnect_delay": "10s", "min_reconnect_delay": "10ms", "tls_ca_cert_file": "/path/to/tls/ca.pem", "tls_cert_file": "/path/to/tls/cert.pem", "tls_key_file": "/path/to/tls/key.pem"

```
    }
  }
```

Create or update a TCP stream handler

The `/tcp-stream-handlers/:tcp-stream-handler` API endpoint provides HTTP PUT access to create or update a specific `:tcp-stream-handler` definition, by handler name.

Example

In the following example, an HTTP PUT request is submitted to the `/tcp-stream-handlers/:tcp-stream-handler` API endpoint to create the handler `incident_log`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "api_version": "pipeline/v1",
  "type": "TCPStreamHandler",
  "metadata": {
    "name": "incident_log"
  },
  "spec": {
    "address": "127.0.0.1:4242",
    "max_connections": 10,
    "min_reconnect_delay": "10ms",
    "max_reconnect_delay": "10s"
  }
}' \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers/incident_log
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/tcp-stream-handlers/:tcp-stream-handler (PUT)	
description	Creates or updates the specified Sensu TCP stream handler.
example URL	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers/incident_log
payload	{ "api_version": "pipeline/v1", "type": "TCPStreamHandler", "metadata": { "name": "incident_log" }, "spec": { "address": "127.0.0.1:4242", "max_connections": 10, "min_reconnect_delay": "10ms", "max_reconnect_delay": "10s" } }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a TCP stream handler

The `/tcp-stream-handlers/:tcp-stream-handler` API endpoint provides HTTP DELETE access to delete a TCP stream handler from Sensu (specified by the handler name).

Example

The following example shows a request to the `/tcp-stream-handlers/:tcp-stream-handler` API endpoint to delete the TCP stream handler `incident_log`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
http://127.0.0.1:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers/incident_log \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/tcp-stream-handlers/:tcp-stream-handler (DELETE)	
description	Removes the specified TCP stream handler from Sensu.
example url	http://hostname:8080/api/enterprise/pipeline/v1/namespaces/default/tcp-stream-handlers/incident_log
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

enterprise/prune/v1alpha

***COMMERCIAL FEATURE**: Access pruning via `enterprise/prune/v1alpha` API endpoints in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: The `enterprise/prune/v1alpha` API endpoints are an alpha feature and may include breaking changes.*

The pruning operation follows the role-based access control (RBAC) permissions of the current user. For example, to prune resources in the `dev` namespace, the current user who sends the prune command must have delete access to the `dev` namespace.

Requests to `enterprise/prune/v1alpha` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.

Create a new pruning command

The `/prune/v1alpha` API endpoint provides HTTP POST access to create a pruning command to delete resources that are not specified in the request body.

Example

In the following example, an HTTP POST request is submitted to the `/prune/v1alpha` API endpoint to create a pruning command for the checks specified in the request body in the `dev` namespace created by any user:

```
curl -X POST \
http://127.0.0.1:8080/api/enterprise/prune/v1alpha\?types\=core/v2.CheckConfig\&allUs
ers\=true\&namespaces\=dev \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
```

```
-d '{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "name": "check-echo",
  "namespace": "dev",
  "labels": {
    "region": "us-west-2",
    "sensu.io/managed_by": "sensuctl"
  },
  "created_by": "admin"
}'
```

The request returns a successful `HTTP/1.1 201 Created` response and a list of the resources that were pruned:

```
[
  {
    "type": "CheckConfig",
    "api_version": "core/v2",
    "name": "check-echo",
    "namespace": "dev",
    "labels": {
      "region": "us-west-2",
      "sensu.io/managed_by": "sensuctl"
    },
    "created_by": "admin"
  }
]
```

API Specification

/prune/v1alpha (POST)	
description	Creates a pruning command to delete the specified resources.
example URL	http://hostname:8080/api/enterprise/prune/v1alpha? types=core/v2.CheckConfig&allUsers=true&namespaces=dev? types=core/v2.CheckConfig&allUsers=true&namespaces=dev

example payload

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "name": "check-echo",
  "namespace": "dev",
  "labels": {
    "region": "us-west-2",
    "sensu.io/managed_by": "sensuctl"
  },
  "created_by": "admin"
}
```

query parameters

- `type`: The fully-qualified name of the resource you want to prune. Example: `?type=core/v2.CheckConfig`.
- `allUsers`: Prune resources created by all users. Mutually exclusive with the `users` parameter. Defaults to false. Example: `?allUsers=true`.
- `clusterWide`: Prune any cluster-wide (non-namespaced) resources that are not defined in the configuration. Defaults to false. Example: `?clusterWide=true`.
- `dryRun`: Print the resources that will be pruned but does not actually delete them. Defaults to false. Example: `?dryRun=true`.
- `labelSelector`: Prune only resources that match the specified labels (accepts multiple values). Labels are a commercial feature. Example: `?labelSelector=[...]`.
- `namespaces`: The namespace where you want to apply pruning. Example: `?namespaces=dev`.
- `users`: Prune only resources that were created by the specified users (accepts multiple values). Defaults to the currently configured sensuctl user. Example: `?users=admin`.

To use multiple values for the parameters that allow them, you must specify the parameter multiple times (for example, `?users=admin&users=dev`) rather than using a comma-separated

list.

output

```
[
  {
    "type": "CheckConfig",
    "api_version": "core/v2",
    "name": "check-echo",
    "namespace": "dev",
    "labels": {
      "region": "us-west-2",
      "sensu.io/managed_by": "sensuctl"
    },
    "created_by": "admin"
  }
]
```

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

enterprise/searches/v1

***COMMERCIAL FEATURE**: Access saved searches in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/searches/v1` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all searches

The `/searches` API endpoint provides HTTP GET access to the list of saved searches.

Example

The following example demonstrates a GET request to the `/search` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the search definitions in the `default` namespace:

```
[
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "incidents-us-west",
      "namespace": "default"
```

```
    },
    "spec": {
      "parameters": [
        "labelSelector:region == \"us-west-1\"",
        "status:incident"
      ],
      "resource": "core.v2/Event"
    }
  },
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "silenced-events",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "silenced:true"
      ],
      "resource": "core.v2/Event"
    }
  },
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "web-agent",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "class:agent",
        "subscription:web"
      ],
      "resource": "core.v2/Entity"
    }
  }
]
```

API Specification

/searches (GET)	
description	Returns the list of saved searches.
example url	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "incidents-us-west",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "labelSelector:region == \"us-west-1\"",
        "status:incident"
      ],
      "resource": "core.v2/Event"
    }
  },
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "silenced-events",
      "namespace": "default"
    },
```

```
    "spec": {
      "parameters": [
        "silenced:true"
      ],
      "resource": "core.v2/Event"
    }
  },
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "web-agent",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "class:agent",
        "subscription:web"
      ],
      "resource": "core.v2/Entity"
    }
  }
]
```

Get a specific search

The `/searches/:search` API endpoint provides HTTP GET access to a specific `:search` definition, by the saved search `name`.

Example

The following example queries the `/searches/:search` API endpoint for the `:search` named `silenced-events`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events \
```

```
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:search` definition (in this example, `silenced-events`):

```
{
  "type": "Search",
  "api_version": "searches/v1",
  "metadata": {
    "name": "silenced-events",
    "namespace": "default"
  },
  "spec": {
    "parameters": [
      "silenced:true"
    ],
    "resource": "core.v2/Event"
  }
}
```

API Specification

/searches/:search (GET)	
description	Returns the specified search.
example url	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	

```
{
  "type": "Search",
  "api_version": "searches/v1",
  "metadata": {
    "name": "silenced-events",
    "namespace": "default"
  },
  "spec": {
    "parameters": [
      "silenced:true"
    ],
    "resource": "core.v2/Event"
  }
}
```

Create or update a search

The `/searches/:search` API endpoint provides HTTP PUT access to create or update a saved search by the saved search `name`.

Example

In the following example, an HTTP PUT request is submitted to the `/searches/:search` API endpoint to create or update a saved search for events that are silenced. The request includes the saved search definition in the request body.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "Search",
  "api_version": "searches/v1",
  "metadata": {
    "name": "silenced-events",
    "namespace": "default"
  },
  "spec": {
```

```
    "parameters": [
      "silenced:true"
    ],
    "resource": "core.v2/Event"
  }
}' \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/searches/:search (PUT)	
description	Creates or updates the specified saved search.
example URL	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
payload	{ "type": "Search", "api_version": "searches/v1", "metadata": { "name": "silenced-events", "namespace": "default" }, "spec": { "parameters": ["silenced:true"], "resource": "core.v2/Event" } }
response codes	¬ **Success**: 201 (Created)

- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a search

The `/searches/:search` API endpoint provides HTTP DELETE access to delete a saved search from Sensu (specified by the saved search name).

Example

The following example shows a request to the `/searches/:search` API endpoint to delete the saved search `silenced-events`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
```

API Specification

/searches/:search (DELETE)	
description	Removes a saved search from Sensu (specified by the search name).
example url	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
response codes	- **Success**: 204 (No Content) - **Missing**: 404 (Not Found) - **Error**: 500 (Internal Server Error)

enterprise/secrets/v1

***COMMERCIAL FEATURE**: Access secrets management in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/secrets/v1` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all secrets providers

The `/providers` API endpoint provides HTTP GET access to a list of secrets providers.

Example

The following example demonstrates a GET request to the `/providers` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the secrets provider definitions:

```
[
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "my_vault",
      "created_by": "admin"
```

```
    },
    "spec": {
      "client": {
        "address": "https://vaultserver.example.com:8200",
        "token": "VAULT_TOKEN",
        "version": "v1",
        "tls": {
          "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
        },
        "max_retries": 2,
        "timeout": "20s",
        "rate_limiter": {
          "limit": 10.0,
          "burst": 100
        }
      }
    }
  }
]
```

***NOTE**: In addition to the `VaultProvider` type, enterprise/secrets/v1 API also includes the `Env` secrets provider type that can retrieve backend environment variables as secrets. Learn more in the secrets providers reference.*

API Specification

/providers (GET)	
description	Returns the list of secrets providers.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers
query parameters	`types` : Defines which type of secrets provider to retrieve. Join with `&` to retrieve multiple types: `?types=Env&types=VaultProvider` .
response filtering	This endpoint supports API response filtering.
response type	Array

response codes

- **Success**: 200 (OK)
- **Error**: 500 (Internal Server Error)

---

output

```
[
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "my_vault",
      "created_by": "admin"
    },
    "spec": {
      "client": {
        "address": "https://vaultserver.example.com:8200",
        "token": "VAULT_TOKEN",
        "version": "v1",
        "tls": {
          "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
        },
        "max_retries": 2,
        "timeout": "20s",
        "rate_limiter": {
          "limit": 10.0,
          "burst": 100
        }
      }
    }
  }
]
```

Get a specific secrets provider

The `/providers/:provider` API endpoint provides HTTP GET access to data for a specific secrets `:provider`, by provider name.

Example

The following example queries the `/providers/:provider` API endpoint for the requested `:provider`, `my_vault`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers/my_vault \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:provider` definition (in this example, `my_vault`):

```
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "my_vault",
    "created_by": "admin"
  },
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
```

API Specification

/providers/:provider (GET)	
description	Returns the specified secrets provider.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers/my_vault
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{ "type": "VaultProvider", "api_version": "secrets/v1", "metadata": { "name": "my_vault", "created_by": "admin" }, "spec": { "client": { "address": "https://vaultserver.example.com:8200", "token": "VAULT_TOKEN", "version": "v1", "tls": { "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem" }, "max_retries": 2, "timeout": "20s", "rate_limiter": { "limit": 10.0, "burst": 100 }

```
      }
    }
  }
```

Create or update a secrets provider

The `/providers/:provider` API endpoint provides HTTP PUT access to create or update a specific `:provider`, by provider name.

Example

The following example demonstrates a request to the `/providers/:provider` API endpoint to update the provider `my_vault`:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "my_vault"
  },
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
```

```
    }
  }
}' \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers/my_vault
```

The request will return a successful `HTTP/1.1 201 Created` response and the complete definition for the provider you created or updated.

API Specification

/providers/:provider (PUT)	
description	Creates or updates the specified secrets provider. The provider resource and API version cannot be altered.
example URL	http://hostname:8080/api/enterprise/secrets/v1/providers/my_vault
payload	<pre>{ "type": "VaultProvider", "api_version": "secrets/v1", "metadata": { "name": "my_vault" }, "spec": { "client": { "address": "https://vaultserver.example.com:8200", "token": "VAULT_TOKEN", "version": "v1", "tls": { "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem" }, "max_retries": 2, "timeout": "20s", "rate_limiter": { "limit": 10.0, "burst": 100</pre>

```
        }
      }
    }
  }
```

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a secrets provider

The `/providers/:provider` API endpoint provides HTTP DELETE access to delete the specified provider from Sensu.

Example

The following example shows a request to the `/providers/:provider` API endpoint to delete the provider `my_vault`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers/my_vault
```

API Specification

/providers/:provider (DELETE)	
description	Deletes the specified provider from Sensu.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers/my_vault

response codes

- **Success**: 204 (No Content)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get a subset of secrets providers with response filtering

The `/providers` API endpoint supports response filtering for a subset of secrets providers data based on labels and the `provider.name` field.

Example

The following example demonstrates a request to the `/providers` API endpoint with response filtering for only secrets provider definitions whose name includes `vault`:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers -G \
--data-urlencode 'fieldSelector=provider.name matches vault'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only provider definitions whose names include `vault`:


```
[
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "vault_dev",
      "created_by": "admin"
    },
    "spec": {
      "client": {
        "address": "http://localhost:8200",
        "agent_address": "",
        "max_retries": 2,
```

```
        "rate_limiter": {
          "burst": 100,
          "limit": 10
        },
        "timeout": "20s",
        "tls": null,
        "token": "\\u003croot_token\\u003e",
        "version": "v2"
      }
    }
  },
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "my_vault",
      "created_by": "admin"
    },
    "spec": {
      "client": {
        "address": "https://vaultserver.example.com:8200",
        "token": "VAULT_TOKEN",
        "version": "v1",
        "tls": {
          "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
        },
        "max_retries": 2,
        "timeout": "20s",
        "rate_limiter": {
          "limit": 10.0,
          "burst": 100
        }
      }
    }
  }
]
```


***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/providers (GET) with response filters	
description	Returns the list of secrets providers that match the response filters applied in the API request.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	[{ "type": "VaultProvider", "api_version": "secrets/v1", "metadata": { "name": "vault_dev", "created_by": "admin" }, "spec": { "client": { "address": "http://localhost:8200", "agent_address": "", "max_retries": 2, "rate_limiter": { "burst": 100, "limit": 10 }, "timeout": "20s", "tls": null, "token": "\\u003croot_token\\u003e", "version": "v2"

```
      }
    }
  },
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "my_vault",
      "created_by": "admin"
    },
    "spec": {
      "client": {
        "address":
"https://vaultserver.example.com:8200",
        "token": "VAULT_TOKEN",
        "version": "v1",
        "tls": {
          "ca_cert":
"/etc/ssl/certs/vault_ca_cert.pem"
        },
        "max_retries": 2,
        "timeout": "20s",
        "rate_limiter": {
          "limit": 10.0,
          "burst": 100
        }
      }
    }
  }
]
```

Get all secrets

The `/secrets` API endpoint provides HTTP GET access to a list of secrets.

Example

The following example demonstrates a GET request to the `/secrets` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets \
-H "Authorization: Key $SENSU_API_KEY"
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the secret definitions in the `default` namespace:

```
[
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sensu-ansible-token",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/ansible#token",
      "provider": "ansible_vault"
    }
  }
]
```

API Specification

/secrets (GET)	
description	Returns the list of secrets for the specified namespace.
example url	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets
response filtering	This endpoint supports API response filtering.
response type	Array

response codes

- **Success**: 200 (OK)
- **Error**: 500 (Internal Server Error)

---

output

```
[
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sensu-ansible-token",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/ansible#token",
      "provider": "ansible_vault"
    }
  }
]
```

Get a specific secret

The `/secrets/:secret` API endpoint provides HTTP GET access to data for a specific `secret`, by secret name.

Example

The following example queries the `/secrets/:secret` API endpoint for the requested `:secret`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:secret` definition (in this example, `sensu-ansible-token`):

```
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "id": "secret/ansible#token",
    "provider": "ansible_vault"
  }
}
```

API Specification

/secrets/:secret (GET)	
description	Returns the specified secret.
example url	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	`{ "type": "Secret", "api_version": "secrets/v1", "metadata": {`

```
    "name": "sensu-ansible-token",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "id": "secret/ansible#token",
    "provider": "ansible_vault"
  }
}
```

Create or update a secret

The `/secrets/:secret` API endpoint provides HTTP PUT access to create or update a specific `secret`, by secret name.

Example

The following example demonstrates a request to the `/secrets/:secret` API endpoint to update the secret `sensu-ansible-token`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default"
  },
  "spec": {
    "id": "secret/ansible#token",
    "provider": "ansible_vault"
  }
}' \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-
```

```
ansible-token
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/secrets/:secret (PUT)	
description	Creates or updates the specified secret.
example URL	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token
payload	{ "type": "Secret", "api_version": "secrets/v1", "metadata": { "name": "sensu-ansible-token", "namespace": "default" }, "spec": { "id": "secret/ansible#token", "provider": "ansible_vault" } }
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete a secret

The `/secrets/:secret` API endpoint provides HTTP DELETE access to delete the specified secret

from Sensu.

Example

The following example shows a request to the `/secrets/:secret` API endpoint to delete the secret `sensu-ansible-token`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token
```

API Specification

/secrets/:secret (DELETE)	
description	Deletes the specified secret from Sensu.
example url	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Get a subset of secrets with response filtering

The `/secrets` API endpoint supports response filtering for a subset of secrets data based on labels and the following fields:

- `secret.name`
- `secret.namespace`

- `secret.provider`
- `secret.id`

Example

The following example demonstrates a request to the `/secrets` API endpoint with response filtering, resulting in a JSON array that contains only secrets definitions for the `vault` provider.

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/enterprise/secrets/v1/secrets -G \
--data-urlencode 'fieldSelector=secret.provider == vault'
```

The example request will result in a successful `HTTP/1.1 200 OK` response and a JSON array that contains only secret definitions for the `vault` provider:


```
[
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "pagerduty_key",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/pagerduty#key",
      "provider": "vault"
    }
  },
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sensu-ansible",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
```

```
      "id": "secret/database#password",
      "provider": "vault"
    }
  },
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sumologic_url",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/sumologic#key",
      "provider": "vault"
    }
  }
]
```

***NOTE**: Read API response filtering for more filter statement examples that demonstrate how to filter responses using different operators with label and field selectors.*

API Specification

/secrets (GET) with response filters	
description	Returns the list of secrets that match the response filters applied in the API request.
example url	http://hostname:8080/api/enterprise/secrets/v1/secrets
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "pagerduty_key",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/pagerduty#key",
      "provider": "vault"
    }
  },
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sensu-ansible",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/database#password",
      "provider": "vault"
    }
  },
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sumologic_url",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/sumologic#key",
      "provider": "vault"
    }
  }
]
```

enterprise/store/v1

***COMMERCIAL FEATURE**: Access the datastore feature in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/store/v1` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get all datastore providers

The `/provider` API endpoint provides HTTP GET access to Sensu datastore data.

Example

The following example demonstrates a GET request to the `/provider` API endpoint, resulting in a JSON map that contains a list of Sensu datastore providers.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/store/v1/provider
-H "Authorization: Key $SENSU_API_KEY" \
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the datastore provider definitions:

```
[
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-other-postgres",
```

```
      "created_by": "admin"
    },
    "spec": {
      "batch_buffer": 0,
      "batch_size": 1,
      "batch_workers": 0,
      "dsn": "postgresql://user:secret@host:port/otherdbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true,
      "enable_round_robin": true
    }
  },
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-postgres",
      "created_by": "admin"
    },
    "spec": {
      "dsn": "postgresql://user:secret@host:port/dbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true,
      "enable_round_robin": true
    }
  }
]
```

API Specification

/provider (GET)	
description	Returns the list of datastore providers.
example url	http://hostname:8080/api/enterprise/store/v1/provider

response type	Map

response codes

- **Success**: 200 (OK)
- **Error**: 500 (Internal Server Error)

output


```
[
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-postgres",
      "created_by": "admin"
    },
    "spec": {
      "batch_buffer": 0,
      "batch_size": 1,
      "batch_workers": 0,
      "dsn":
"postgresql://user:secret@host:port/otherdbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true,
      "enable_round_robin": true
    }
  },
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-postgres",
      "created_by": "admin"
    },
    "spec": {
      "dsn": "postgresql://user:secret@host:port/dbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true,
```

```
        "enable_round_robin": true
      }
    }
  ]
```

Get a specific datastore provider

The `/provider/:provider` API endpoint provides HTTP GET access to retrieve a Sensu datastore provider.

Example

The following example queries the `/provider/:provider` API endpoint for a specific `:provider`:

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:provider` definition (in this example, `my-postgres`):

```
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres",
    "created_by": "admin"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
```

```
    "pool_size": 20,
    "strict": true,
    "enable_round_robin": true
  }
}
```

API Specification

/provider/:provider (GET)	
description	Returns the specified datastore provider.
example url	http://hostname:8080/api/enterprise/store/v1/provider/my-postgres
url parameters	Required: `my-postgres` (name of provider to retrieve).
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	<pre>{ "type": "PostgresConfig", "api_version": "store/v1", "metadata": { "name": "my-postgres", "created_by": "admin" }, "spec": { "batch_buffer": 0, "batch_size": 1, "batch_workers": 0, "dsn": "postgresql://user:secret@host:port/dbname", "max_conn_lifetime": "5m", "max_idle_conns": 2, "pool_size": 20, "strict": true,</pre>

```
    "enable_round_robin": true
  }
}
```

Create or update a datastore provider

The `/provider/:provider` API endpoint provides HTTP PUT access to create or update a Sensu datastore provider.

Example

```
curl -X PUT \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres \
-H "Authorization: Key $SENSU_API_KEY" \
-d '{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true,
    "enable_round_robin": true
  }
}'
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/provider/:provider (PUT)	
description	Creates a datastore provider.
example url	http://hostname:8080/api/enterprise/store/v1/provider/my-postgres
url parameters	Required: `my-postgres` (name to use for provider).
payload	`{ "type": "PostgresConfig", "api_version": "store/v1", "metadata": { "name": "my-postgres" }, "spec": { "batch_buffer": 0, "batch_size": 1, "batch_workers": 0, "dsn": "postgresql://user:secret@host:port/dbname", "max_conn_lifetime": "5m", "max_idle_conns": 2, "pool_size": 20, "strict": true, "enable_round_robin": true } }`
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Delete a datastore provider

The `/provider/:provider` API endpoint provides HTTP DELETE access to remove a Sensu datastore provider.

Example

The following example shows a request to the `/provider/:provider` API endpoint to remove the Sensu datastore provider with the ID `my-postgres`, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres
```

API Specification

/provider/:provider (DELETE)	
description	Removes the specified datastore provider.
example url	http://hostname:8080/api/enterprise/store/v1/provider/my-postgres
url parameters	Required: `my-postgres` (name of provider to delete).
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

enterprise/web/v1

***COMMERCIAL FEATURE**: Access web UI configuration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

***NOTE**: Requests to `enterprise/web/v1` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

Get the web UI configuration

The `/config` API endpoint provides HTTP GET access to the global web UI configuration.

Example

The following example demonstrates a GET request to the `/config` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/web/v1/config \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json'
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON array that contains the web UI configuration definitions:

```
[
  {
    "type": "GlobalConfig",
    "api_version": "web/v1",
    "metadata": {
      "name": "custom-web-ui",
```

```
      "created_by": "admin"
    },
    "spec": {
      "always_show_local_cluster": false,
      "catalog": {
        "disabled": false,
        "release_version": "version",
        "url": "https://catalog.sensu.io"
      },
      "default_preferences": {
        "poll_interval": 120000,
        "page_size": 500,
        "serialization_format": "YAML",
        "theme": "sensu"
      },
      "license_expiry_reminder": "1080h0m0s",
      "link_policy": {
        "allow_list": true,
        "urls": [
          "https://example.com",
          "steamapp://34234234",
          "//google.com",
          "//*.google.com",
          "//bob.local",
          "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
        ]
      },
      "page_preferences": [
        {
          "order": "LASTSEEN",
          "page": "entities",
          "page_size": 50,
          "selector": "proxy in entity.subscriptions"
        },
        {
          "order": "NAME",
          "page": "checks",
          "page_size": 100
        }
      ],
      "signin_message": "with your **LDAP or system credentials**"
```

```
      }
    }
]
```

API Specification

/web (GET)	
description	Returns the list of global web UI configurations.
example url	http://hostname:8080/api/enterprise/web/v1/config
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "type": "GlobalConfig",
    "api_version": "web/v1",
    "metadata": {
      "name": "custom-web-ui",
      "created_by": "admin"
    },
    "spec": {
      "always_show_local_cluster": false,
      "catalog": {
        "disabled": false,
        "release_version": "version",
        "url": "https://catalog.sensu.io"
      },
      "default_preferences": {
        "poll_interval": 120000,
        "page_size": 500,
        "serialization_format": "YAML",
        "theme": "sensu"
```

```
      },
      "license_expiry_reminder": "1080h0m0s",
      "link_policy": {
        "allow_list": true,
        "urls": [
          "https://example.com",
          "steamapp://34234234",
          "//google.com",
          "//*.google.com",
          "//bob.local",
          "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
        ]
      },
      "page_preferences": [
        {
          "order": "LASTSEEN",
          "page": "entities",
          "page_size": 50,
          "selector": "proxy in entity.subscriptions"
        },
        {
          "order": "NAME",
          "page": "checks",
          "page_size": 100
        }
      ],
      "signin_message": "with your **LDAP or system
credentials**"
    }
  }
]
```


Get a specific web UI configuration

The `/config/:globalconfig` API endpoint provides HTTP GET access to global web UI configuration data, specified by configuration name.

Example

The following example queries the `/config/:globalconfig` API endpoint for the `:globalconfig` named `custom-web-ui`:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/web/v1/config/custom-web-ui \
-H "Authorization: Key $SENSU_API_KEY"
```

The request will return a successful `HTTP/1.1 200 OK` response and a JSON map that contains the requested `:globalconfig` definition (in this example, `custom-web-ui`):


```
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui",
    "created_by": "admin"
  },
  "spec": {
    "always_show_local_cluster": false,
    "catalog": {
      "disabled": false,
      "release_version": "version",
      "url": "https://catalog.sensu.io"
    },
    "default_preferences": {
      "poll_interval": 120000,
      "page_size": 500,
      "serialization_format": "YAML",
      "theme": "sensu"
    },
    "license_expiry_reminder": "1080h0m0s",
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
```

```
          "//*.google.com",
          "//bob.local",
          "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
        ]
      },
      "page_preferences": [
        {
          "order": "LASTSEEN",
          "page": "entities",
          "page_size": 50,
          "selector": "proxy in entity.subscriptions"
        },
        {
          "order": "NAME",
          "page": "checks",
          "page_size": 100
        }
      ],
      "signin_message": "with your **LDAP or system credentials**"
    }
  }
```

API Specification

/config/:globalconfig (GET)	
description	Returns the specified global web UI configuration.
example url	http://hostname:8080/api/enterprise/web/v1/config/custom-web-ui
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output


```
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui",
    "created_by": "admin"
  },
  "spec": {
    "always_show_local_cluster": false,
    "catalog": {
      "disabled": false,
      "release_version": "version",
      "url": "https://catalog.sensu.io"
    },
    "default_preferences": {
      "poll_interval": 120000,
      "page_size": 500,
      "serialization_format": "YAML",
      "theme": "sensu"
    },
    "license_expiry_reminder": "1080h0m0s",
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
      ]
    },
    "page_preferences": [
      {
        "order": "LASTSEEN",
        "page": "entities",
        "page_size": 50,
        "selector": "proxy in entity.subscriptions"
      },
      {
```

```
          "order": "NAME",
          "page": "checks",
          "page_size": 100
        }
      ],
      "signin_message": "with your **LDAP or system
credentials**"
    }
}
```

Create and update a web UI configuration

The `/config/:globalconfig` API endpoint provides HTTP PUT access to create and update global web UI configurations, specified by configuration name.

Example

In the following example, an HTTP PUT request is submitted to the `/config/:globalconfig` API endpoint to update the `custom-web-ui` configuration:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui"
  },
  "spec": {
    "always_show_local_cluster": false,
    "catalog": {
      "disabled": false,
      "release_version": "version",
      "url": "https://catalog.sensu.io"
    },
    "default_preferences": {
```

```
      "poll_interval": 120000,
      "page_size": 500,
      "serialization_format": "YAML",
      "theme": "sensu"
    },
    "license_expiry_reminder": "1080h0m0s",
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
      ]
    },
    "page_preferences": [
      {
        "order": "LASTSEEN",
        "page": "entities",
        "page_size": 50,
        "selector": "proxy in entity.subscriptions"
      },
      {
        "order": "NAME",
        "page": "checks",
        "page_size": 100
      }
    ],
    "signin_message": "with your **LDAP or system credentials**"
  }
}' \
http://127.0.0.1:8080/api/enterprise/web/v1/config/custom-web-ui
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/config/:globalconfig (PUT)	
description	Creates or updates the specified global web UI configuration.
example URL	http://hostname:8080/api/enterprise/web/v1/config/custom-web-ui
payload	


```
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui"
  },
  "spec": {
    "always_show_local_cluster": false,
    "catalog": {
      "disabled": false,
      "release_version": "version",
      "url": "https://catalog.sensu.io"
    },
    "default_preferences": {
      "poll_interval": 120000,
      "page_size": 500,
      "serialization_format": "YAML",
      "theme": "sensu"
    },
    "license_expiry_reminder": "1080h0m0s",
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
      ]
    },
    "page_preferences": [
      {
```

```
          "order": "LASTSEEN",
          "page": "entities",
          "page_size": 50,
          "selector": "proxy in entity.subscriptions"
        },
        {
          "order": "NAME",
          "page": "checks",
          "page_size": 100
        }
      ],
      "signin_message": "with your **LDAP or system
credentials**"
    }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a web UI configuration

The `/config/:globalconfig` API endpoint provides HTTP DELETE access to delete a global web UI configuration from Sensu, specified by the configuration name.

Example

The following example shows a request to the `/config/:globalconfig` API endpoint to delete the global web UI configuration named `custom-web-ui`, resulting in a successful `HTTP/1.1 204 No Content` response:

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/web/v1/config/custom-web-ui
```

API Specification

/config/:globalconfig (DELETE)	
description	Removes the specified global web UI configuration from Sensu.
example url	http://hostname:8080/api/enterprise/web/v1/config/custom-web-ui
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Other APIs

In addition to the core/v2 API and enterprise APIs, Sensu offers endpoints for basic authentication, health, license, metrics, and version:

- /auth
- /health
- /license
- /metrics
- /ready
- /version

/auth

Generate an access token and a refresh token

The `/auth` API endpoint provides HTTP GET access to generate an access token and a refresh token using Sensu's basic authentication.

The access and refresh tokens are JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature.

Example

The following example queries the `/auth` API endpoint with a given username and password to determine whether the credentials are valid and retrieve an access token and a refresh token:

```
curl -X GET \
http://127.0.0.1:8080/auth \
-u myusername:mypassword
```

The request results in a successful `HTTP/1.1 200 OK` response to indicate that the credentials are valid, along with an access token and a refresh token:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1544582187,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
```

API Specification

/auth (GET)	
description	Generates an access and a refresh token used for accessing the API using Sensu's basic authentication. Access tokens last for approximately 15 minutes. When your token expires, you should receive a `401 Unauthorized` response from the API. To generate a new access token, use the `/auth/token` API endpoint.
example url	http://hostname:8080/auth
output	`{ "access_token": "eyJhbGciOiJIUzI1NiIs...", "expires_at": 1544582187, "refresh_token": "eyJhbGciOiJIUzI1NiIs..." }`
response codes	¬ **Valid credentials**: 200 (OK) ¬ **Invalid credentials**: 401 (Unauthorized) ¬ **Error**: 500 (Internal Server Error)

Test basic auth user credentials

The `/auth/test` API endpoint provides HTTP GET access to test basic authentication user credentials that were created with Sensu's built-in basic authentication.

***NOTE**: The `/auth/test` endpoint only tests user credentials created with Sensu's built-in basic authentication. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).*

Example

In the following example, querying the `/auth/test` API endpoint with a given username and

password should return an `HTTP/1.1 200 OK` response, indicating that the credentials are valid:

```
curl -X GET \
http://127.0.0.1:8080/auth/test \
-u myusername:mypassword
```

API Specification

/auth/test (GET)	
description	Tests basic authentication credentials (username and password) that were created with Sensu's core/v2/users API.
example url	http://hostname:8080/auth/test
response codes	¬ **Valid credentials**: 200 (OK) ¬ **Invalid credentials**: 401 (Unauthorized) ¬ **Error**: 500 (Internal Server Error)

Renew an access token

The `/auth/token` API endpoint provides HTTP POST access to renew an access token.

Example

In the following example, an HTTP POST request is submitted to the `/auth/token` API endpoint to generate a valid access token. The request includes the refresh token in the request body.

```
curl -X POST \
http://127.0.0.1:8080/auth/token \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
-H 'Content-Type: application/json' \
```

```
  -d '{"refresh_token": "eyJhbGciOiJIUzI1NiIs..."}'
```

The request results in a successful `HTTP/1.1 200 OK` response, along with the new access token:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1544582187,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
```

The access and refresh tokens are JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature.

API Specification

/auth/token (POST)	
description	Generates a new access token using a refresh token and an expired access token.
example url	http://hostname:8080/auth/token
example payload	`{ "refresh_token": "eyJhbGciOiJIUzI1NiIs..." }`
output	`{ "access_token": "eyJhbGciOiJIUzI1NiIs...", "expires_at": 1544582187, "refresh_token": "eyJhbGciOiJIUzI1NiIs..." }`
response codes	

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

/health

Get health data for your Sensu instance

The `/health` API endpoint provides HTTP GET access to health data for your Sensu instance.

Example

The following example demonstrates a GET request to the `/health` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/health
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON map that contains Sensu health data:


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 2882886652148554927,
      "MemberIDHex": "8923110df66458af",
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 4255616344056076734,
    "member_id": 2882886652148554927,
    "raft_term": 26
  },
  "PostgresHealth": [
```

```
    {
      "Name": "my-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```

***NOTE**: If your Sensu instance is not configured to use a PostgreSQL datastore, the health payload will not include `PostgresHealth`.*

API Specification

/health (GET)	
description	Returns health information about the Sensu instance.
example url	http://hostname:8080/health
query parameters	`timeout`: Defines the timeout when querying etcd. Default is `3`.
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 400 (Bad Request) ***NOTE**: The HTTP response codes for the health endpoint indicate whether your request reached Sensu rather than the health of your Sensu instance. To determine the health of your Sensu instance, you must process the JSON response body for your request. The health specification describes each attribute in the response body.*
output	{ "Alarms": null, "ClusterHealth": [

```
    {
      "MemberID": 2882886652148554927,
      "MemberIDHex": "8923110df66458af",
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 4255616344056076734,
    "member_id": 2882886652148554927,
    "raft_term": 26
  },
  "PostgresHealth": [
    {
      "Name": "my-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```


Get health data for your agent transport

The `/health` API endpoint provides HTTP GET access to health data for your Sensu agent transport via the backend WebSocket. Sensu backend `/health` API information is duplicated by this agent transport API endpoint as an affordance to satisfy the load balancing and security requirements of some deployments.

Example

The following example demonstrates a GET request to the backend WebSocket `/health` API endpoint using the default WebSocket port 8081:

```
curl -X GET \
http://127.0.0.1:8081/health
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON map that contains Sensu agent transport status:


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 2882886652148554927,
      "MemberIDHex": "8923110df66458af",
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 4255616344056076734,
    "member_id": 2882886652148554927,
    "raft_term": 26
  },
  "PostgresHealth": [
    {
      "Name": "my-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```


***NOTE**: If your Sensu instance is not configured to use a PostgreSQL datastore, the health payload will not include `PostgresHealth`.*

API Specification

/health (GET)	
description	Returns health information about the Sensu agent transport.

example url	http://hostname:8081/health
query parameters	`timeout`: Defines the timeout when querying etcd. Default is `3`.
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 400 (Bad Request) ***NOTE**: The HTTP response codes for the health endpoint indicate whether your request reached Sensu rather than the health of your Sensu instance. To determine the health of your Sensu instance, you must process the JSON response body for your request. The health specification describes each attribute in the response body.*
output	{ "Alarms": null, "ClusterHealth": [{ "MemberID": 2882866521485549 27, "MemberIDHex": "8923110df66458af", "Name": "default", "Err": "", "Healthy": true }], "Header": { "cluster_id": 4255616344056076734, "member_id": 2882866521485549 27, "raft_term": 26 }, "PostgresHealth": [{ "Name": "my-postgres", "Active": false, "Healthy": false }]

```
}
```

/license

***NOTE**: Requests to `/license` API endpoints require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.*

For more information about commercial features designed for enterprises, read Get started with commercial features.

Get the active license configuration

The `/license` API endpoint provides HTTP GET access to the active license configuration.

Example

The following example demonstrates a GET request to the `/license` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/licensing/v2/license \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json'
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON map that contains the license definition:

```
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
```

```
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
        "algorithm": "PSS",
        "hashAlgorithm": "SHA256",
        "saltLength": 20
      }
    },
    "signature": "XXXXXXXXXX",
    "metadata": {}
  }
}
```

API Specification

/license (GET)	
description	Returns the active commercial license configuration. To download your license, log in to your Sensu account or contact the Sensu sales team for a free trial.
example url	http://hostname:8080/api/enterprise/licensing/v2/license
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output


```
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
        "algorithm": "PSS",
        "hashAlgorithm": "SHA256",
        "saltLength": 20
      }
    },
    "signature": "XXXXXXXXXX",
    "metadata": {}
  }
}
```


Activate a commercial license

The `/license` API endpoint provides HTTP PUT access to activate a commercial license.

***NOTE**: For clustered configurations, you only need to activate your license for one of the backends within the cluster.*

Example

In the following example, an HTTP PUT request is submitted to the `/license` API endpoint to create the license definition:

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
        "algorithm": "PSS",
        "hashAlgorithm": "SHA256",
        "saltLength": 20
      }
    },
    "signature": "XXXXXXXXXX",
```

```
    "metadata": {}
  }
}' \
http://127.0.0.1:8080/api/enterprise/licensing/v2/license
```

The request will return a successful `HTTP/1.1 201 Created` response.

API Specification

/license (PUT)	
description	Activates a commercial license or updates an existing license configuration. To download your license, log in to your Sensu account or contact the Sensu sales team for a free trial.
example url	http://hostname:8080/api/enterprise/licensing/v2/license
payload	

```
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
```

```
          "algorithm": "PSS",
          "hashAlgorithm": "SHA256",
          "saltLength": 20
        }
      },
      "signature": "XXXXXXXXXX",
      "metadata": {}
    }
  }
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a commercial license

The `/license` API endpoint provides HTTP DELETE access to remove a commercial license.

Example

The following example shows a request to the `/license` API endpoint to delete the commercial license, resulting in a successful `HTTP/1.1 204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/enterprise/licensing/v2/license \
-H "Authorization: Key $SENSU_API_KEY"
```

API Specification

/license (DELETE)	
description	Removes the commercial license.

example url	http://hostname:8080/api/enterprise/licensing/v2/license
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

/metrics

Get Sensu metrics

The `/metrics` API endpoint provides HTTP GET access to internal Sensu metrics in Prometheus format, including embedded etcd, memory usage, garbage collection, and gRPC metrics.

Example

The following example demonstrates a GET request to the `/metrics` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/metrics
```

The request results in a successful `HTTP/1.1 200 OK` response and plaintext output that contains internal Sensu metrics:

```
# HELP etcd_debugging_mvcc_compact_revision The revision of the last compaction in
store.
# TYPE etcd_debugging_mvcc_compact_revision gauge
etcd_debugging_mvcc_compact_revision 300
# HELP etcd_debugging_mvcc_current_revision The current revision of store.
# TYPE etcd_debugging_mvcc_current_revision gauge
etcd_debugging_mvcc_current_revision 316
# HELP etcd_debugging_mvcc_db_compaction_keys_total Total number of db keys
compacted.
# TYPE etcd_debugging_mvcc_db_compaction_keys_total counter
etcd_debugging_mvcc_db_compaction_keys_total 274
# HELP etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds Bucketed
histogram of db compaction pause duration.
# TYPE etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds histogram
etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds_bucket{le="1"} 0
etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds_bucket{le="2"} 0
```

```
...
```

API Specification

/metrics (GET)	
description	Returns internal Sensu metrics in Prometheus format, including embedded etcd, memory usage, garbage collection, and gRPC metrics.
example url	http://hostname:8080/metrics
response type	Prometheus-formatted plaintext
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	`# HELP etcd_debugging_mvcc_compact_revision The revision of the last compaction in store.` `# TYPE etcd_debugging_mvcc_compact_revision gauge` `etcd_debugging_mvcc_compact_revision 300` `# HELP etcd_debugging_mvcc_current_revision The current revision of store.` `# TYPE etcd_debugging_mvcc_current_revision gauge` `etcd_debugging_mvcc_current_revision 316` `# HELP etcd_debugging_mvcc_db_compaction_keys_total Total number of db keys compacted.` `# TYPE etcd_debugging_mvcc_db_compaction_keys_total counter` `etcd_debugging_mvcc_db_compaction_keys_total 274` `# HELP etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds Bucketed histogram of db compaction pause duration.` `# TYPE etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds histogram` `etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds_bucket{le="1"} 0` `etcd_debugging_mvcc_db_compaction_pause_duration_millisecon`

```
ds_bucket{le="2"} 0
...
```

/ready

Get API readiness data for your Sensu instance

The `/ready` API endpoint provides HTTP GET access to information about whether your Sensu instance is ready to serve API requests.

Example

The following example demonstrates a GET request to the `/ready` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/ready
```

The request results in a successful `HTTP/1.1 200 OK` response and a text response body:

```
ready
```

If the backend configuration includes an `api-serve-wait-time` duration, the request will result in an `HTTP/1.1 503 Service Unavailable` response. Until the `api-serve-wait-time` duration expires, the text response body will state that the API is unavailable:

```
API unavailable during startup.
See api-serve-wait-time settings.
```

***NOTE**: `503 Service Unavailable` responses include a `Retry-After` header that lists the specified `api-serve-wait-time` duration.*

API Specification

/ready (GET)	
description	Returns information about whether the Sensu instance is ready to serve API requests.
example url	http://hostname:8080/ready
response type	text
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 503 (Service Unavailable)
output	200 (OK): `ready` 503 (Service Unavailable): `API unavailable during startup.` `See api-serve-wait-time settings.`

Get agent connection readiness data for your Sensu instance

The `/ready` agent transport API endpoint provides HTTP GET access to information about whether your Sensu agent transport is ready to accept agent WebSocket connections.

Example

The following example demonstrates a GET request to the backend agent transport `/ready` endpoint

using the default agent listener port 8081:

```
curl -X GET \
http://127.0.0.1:8081/ready
```

The request results in a successful `HTTP/1.1 200 OK` response and a text response body:

```
ready
```

If the backend configuration includes an `agent-serve-wait-time` duration, the request will result in an `HTTP/1.1 503 Service Unavailable` response.Until the `agent-serve-wait-time` duration expires, the text response body will state that agentd is unavailable:

```
agentd temporarily unavailable during startup
```

***NOTE**: `503 Service Unavailable` responses include a `Retry-After` header that lists the specified `agent-serve-wait-time` duration.*

API Specification

/ready (GET)	
description	Returns information about whether the Sensu instance is ready to accept agent connections.
example url	http://hostname:8081/ready
response type	text
response codes	┐ **Success**: 200 (OK) ┐ **Error**: 503 (Service Unavailable)

output

200 (OK):

```
ready
```

503 (Service Unavailable):

```
agentd temporarily unavailable during startup
```

/version

Get the Sensu backend and etcd versions

The `/version` API endpoint provides HTTP GET access to the Sensu backend and etcd versions for the Sensu instance.

Example

The following example demonstrates a GET request to the `/version` API endpoint:

```
curl -X GET \
http://127.0.0.1:8080/version
```

The request results in a successful `HTTP/1.1 200 OK` response and a JSON map that contains Sensu version data:

```
{
  "etcd": {
    "etcdserver": "3.5.0",
    "etcdcluster": "3.5.0"
  },
  "sensu_backend": "6.4.0"
}
```

API Specification

/version (GET)	
description	Returns the etcd server version and Sensu backend version. For

	clustered Sensu installations with the default embedded etcd, also returns the etcd cluster version (which may not match the etcd server version or the cluster versions of other backends in the cluster).
example url	http://hostname:8080/version
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	{ "etcd": { "etcdserver": "3.5.0", "etcdcluster": "3.5.0" }, "sensu_backend": "6.4.0" }

Reference Index

This index links to every reference in the Sensu documentation.Reference documentation includes specifications, examples, configuration notes, and other details about Sensu resources, the agent and backend, and Sensu query expressions.

- Plugins
- Ready
- Role-based access control (RBAC)
- Rule templates
- Searches
- Secrets
- Secrets providers
- Sensu query expressions
- Service components
- Silencing
- Subscriptions
- Sumo Logic metrics handlers
- TCP stream handlers
- Tessen
- Tokens
- Web UI configuration

Plugins

Sensu plugins provide executable scripts or other programs that you can use as Sensu checks, handlers, and mutators.

Plugins are service-specific and have different setup and configuration requirements. Many Sensu-supported plugins include quick-start templates that you only need to edit to match your configuration. Each plugin has self-contained documentation with in-depth information about how to install and use it.

Find Sensu plugins

Use the Sensu Catalog to find and enable many plugins directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.

To find many more available Sensu plugins, search Bonsai, the Sensu asset hub. Bonsai lists hundreds of Sensu plugins with installation instructions and usage examples.

We also list popular Sensu plugins in the featured integrations section.

Write your own custom plugins

Write your own Sensu plugins in almost any programming language with Sensu's plugin specification. The Sensu Go plugin SDK library provides a framework for building Sensu Go plugins so that all you need to do is define plugin arguments and input validation and execution functions.

If you are interested in sharing your plugin with other Sensu users, you can find guidance for contributing plugins, pinning versions, writing plugin READMEs, and transferring repos to community responsibility at the Sensu plugins community GitHub repo

Use Nagios plugins

The Sensu plugin specification is compatible with the Nagios plugin specification, so you can use Nagios plugins with Sensu without any modification. Sensu allows you to bring new life to the 50+

plugins in the official Nagios Plugins project, a mature source of monitoring plugins, and more than 4000 plugins in the Nagios Exchange.

Assets reference

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. You can use dynamic runtime assets to provide the plugins, libraries, and runtimes you need to automate your monitoring workflows. Sensu supports dynamic runtime assets for checks, filters, mutators, and handlers.

Use the Sensu Catalog to find, configure, and install many dynamic runtime assets directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.

You can also discover, download, and share dynamic runtime assets using Bonsai, the Sensu asset hub. Read Use assets to install plugins to get started.

> ***NOTE**: Dynamic runtime assets are not required to use Sensu Go. You can install Sensu plugins using the sensu-install tool or a configuration management solution.*

The Sensu backend executes handler, filter, and mutator dynamic runtime assets. The Sensu agent executes check dynamic runtime assets. At runtime, the backend or agent sequentially evaluates dynamic runtime assets that appear in the `runtime_assets` attribute of the handler, filter, mutator, or check being executed.

Dynamic runtime asset example (minimum required attributes)

This example shows a dynamic runtime asset resource definition that includes the minimum required attributes:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: check_script
spec:
```

```
  builds:
  - sha512:
4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4a8
7450576b2c58ad9ab40c9e2edc31b288d066b195b21b
    url: http://example.com/asset.tar.gz
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_script"
  },
  "spec": {
    "builds": [
      {
        "url": "http://example.com/asset.tar.gz",
        "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4a
87450576b2c58ad9ab40c9e2edc31b288d066b195b21b"
      }
    ]
  }
}
```

Install location for dynamic runtime assets

If you use a Sensu package, dynamic runtime assets are installed at `/var/cache`.

If you use a Sensu Docker image, dynamic runtime assets are installed at `/var/lib`.

Dynamic runtime asset builds

A dynamic runtime asset build is the combination of an artifact URL, SHA512 checksum, and optional Sensu query expression filters.Each asset definition may describe one or more builds.

***NOTE**: Dynamic runtime assets that provide `url` and `sha512` attributes at the top level of the `spec` scope are single-build assets, and this form of asset defintion is deprecated. We recommend using multiple-build asset defintions, which specify one or more `builds` under the `spec` scope.*

Asset example: Multiple builds

This example shows the resource definition for the sensu/check-cpu-usage dynamic runtime asset, which has multiple builds:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: check-cpu-usage
  labels:
  annotations:
    io.sensu.bonsai.url: https://bonsai.sensu.io/assets/sensu/check-cpu-usage
    io.sensu.bonsai.api_url: https://bonsai.sensu.io/api/v1/assets/sensu/check-cpu-
usage
    io.sensu.bonsai.tier: Community
    io.sensu.bonsai.version: 0.2.2
    io.sensu.bonsai.namespace: sensu
    io.sensu.bonsai.name: check-cpu-usage
    io.sensu.bonsai.tags: ''
spec:
  builds:
  - url:
https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_windows_amd64.tar.gz
    sha512:
900cfdf28d6088b929c4bf9a121b628971edee5fa5cbc91a6bc1df3bd9a7f8adb1fcfb7b1ad70589ed5b
4f5ec87d9a9a3ba95bcf2acda56b0901406f14f69fe7
    filters:
    - entity.system.os == 'windows'
    - entity.system.arch == 'amd64'
  - url:
https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_darwin_amd64.tar.gz
    sha512:
```

```
db81ee70426114e4cd4b3f180f2b0b1e15b4bffc09d7f2b41a571be2422f4399af3fbd2fa2918b883190
9ab4bc2d3f58d0aa0d7b197d3a218b2391bb5c1f6913
    filters:
    - entity.system.os == 'darwin'
    - entity.system.arch == 'amd64'
  - url:
https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_armv7.tar.gz
    sha512:
400aacce297176e69f3a88b0aab0ddfdbe9dd6a37a673cb1774c8d4750a91cf7713a881eef26ea21d200
f74cb20818161c773490139e6a6acb92cbd06dee994c
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'armv7'
  - url:
https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_arm64.tar.gz
    sha512:
bef7802b121ac2a2a5c5ad169d6003f57d8b4f5e83eae998a0e0dd1e7b89678d4a62e678d153edacdd65
fd1d0123b5f51308622690455e77cec6deccfa183397
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'arm64'
  - url:
https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_386.tar.gz
    sha512:
a2dcb5324952567a61d76a2e331c1c16df69ef0e0b9899515dad8d1531b204076ad0c008f59fc2f4735a
5a779afb0c1baa132268c41942b203444e377fe8c8e5
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == '386'
  - url:
https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_amd64.tar.gz
    sha512:
24539739b5eb19bbab6eda151d0bcc63a0825afdfef3bc1ec3670c7b0a00fbbb2fd006d605a7a038b322
69a22026d8947324f2bc0acdf35e8563cf4cb8660d7f
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
```

JSON


```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-cpu-usage",
    "labels": null,
    "annotations": {
      "io.sensu.bonsai.url": "https://bonsai.sensu.io/assets/sensu/check-cpu-usage",
      "io.sensu.bonsai.api_url": "https://bonsai.sensu.io/api/v1/assets/sensu/check-cpu-usage",
      "io.sensu.bonsai.tier": "Community",
      "io.sensu.bonsai.version": "0.2.2",
      "io.sensu.bonsai.namespace": "sensu",
      "io.sensu.bonsai.name": "check-cpu-usage",
      "io.sensu.bonsai.tags": ""
    }
  },
  "spec": {
    "builds": [
      {
        "url": "https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_windows_amd64.tar.gz",
        "sha512": "900cfdf28d6088b929c4bf9a121b628971edee5fa5cbc91a6bc1df3bd9a7f8adb1fcfb7b1ad70589ed5b4f5ec87d9a9a3ba95bcf2acda56b0901406f14f69fe7",
        "filters": [
          "entity.system.os == 'windows'",
          "entity.system.arch == 'amd64'"
        ]
      },
      {
        "url": "https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-usage_0.2.2_darwin_amd64.tar.gz",
        "sha512": "db81ee70426114e4cd4b3f180f2b0b1e15b4bffc09d7f2b41a571be2422f4399af3fbd2fa2918b8831909ab4bc2d3f58d0aa0d7b197d3a218b2391bb5c1f6913",
        "filters": [
          "entity.system.os == 'darwin'",
```

```
          "entity.system.arch == 'amd64'"
        ]
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_armv7.tar.gz",
        "sha512":
"400aacce297176e69f3a88b0aab0ddfdbe9dd6a37a673cb1774c8d4750a91cf7713a881eef26ea21d20
0f74cb20818161c773490139e6a6acb92cbd06dee994c",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'armv7'"
        ]
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_arm64.tar.gz",
        "sha512":
"bef7802b121ac2a2a5c5ad169d6003f57d8b4f5e83eae998a0e0dd1e7b89678d4a62e678d153edacdd6
5fd1d0123b5f51308622690455e77cec6deccfa183397",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'arm64'"
        ]
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
usage_0.2.2_linux_386.tar.gz",
        "sha512":
"a2dcb5324952567a61d76a2e331c1c16df69ef0e0b9899515dad8d1531b204076ad0c008f59fc2f4735
a5a779afb0c1baa132268c41942b203444e377fe8c8e5",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == '386'"
        ]
      },
      {
        "url":
"https://assets.bonsai.sensu.io/a7ced27e881989c44522112aa05dd3f25c8f1e49/check-cpu-
```

```
usage_0.2.2_linux_amd64.tar.gz",
        "sha512":
"24539739b5eb19bbab6eda151d0bcc63a0825afdfef3bc1ec3670c7b0a00fbbb2fd006d605a7a038b32
269a22026d8947324f2bc0acdf35e8563cf4cb8660d7f",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ]
      }
    ]
  }
}
```

Asset example: Single build (deprecated)

This example shows the resource definition for a dynamic runtime asset with a single build:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: check_cpu_linux_amd64
  labels:
    origin: bonsai
  annotations:
    project_url: https://bonsai.sensu.io/assets/asachs01/sensu-go-cpu-check
    version: 0.0.3
spec:
  url:
https://assets.bonsai.sensu.io/981307deb10ebf1f1433a80da5504c3c53d5c44f/sensu-go-
cpu-check_0.0.3_linux_amd64.tar.gz
  sha512:
487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a72b786ef0ca396a0a12c8d006
ac7fa21923e0e9ae63419a4d56aec41fccb574c1a5d3
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
  headers:
    Authorization: 'Bearer {{ .annotations.asset_token | default "N/A" }}'
```

```
    X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu_linux_amd64",
    "labels": {
      "origin": "bonsai"
    },
    "annotations": {
      "project_url": "https://bonsai.sensu.io/assets/asachs01/sensu-go-cpu-check",
      "version": "0.0.3"
    }
  },
  "spec": {
    "url":
"https://assets.bonsai.sensu.io/981307deb10ebf1f1433a80da5504c3c53d5c44f/sensu-go-
cpu-check_0.0.3_linux_amd64.tar.gz",
    "sha512":
"487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a72b786ef0ca396a0a12c8d00
6ac7fa21923e0e9ae63419a4d56aec41fccb574c1a5d3",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "headers": {
      "Authorization": "Bearer {{ .annotations.asset_token | default \"N/A\" }}",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
}
```

Dynamic runtime asset build evaluation

For each build provided in a dynamic runtime asset, Sensu will evaluate any defined filters to determine whether any build matches the agent or backend service's environment. If all filters specified

on a build evaluate to `true`, that build is considered a match. For dynamic runtime assets with multiple builds, only the first build that matches will be downloaded and installed.

Dynamic runtime asset build download

Sensu downloads the dynamic runtime asset build on the host system where the asset contents are needed to execute the requested command. For example, if a check definition references a dynamic runtime asset, the Sensu agent that executes the check will download the asset the first time it executes the check. The dynamic runtime asset build the agent downloads will depend on the filter rules associated with each build defined for the asset.

Sensu backends follow a similar process when pipeline elements (filters, mutators, and handlers) request dynamic runtime asset installation as part of operation.

> ***NOTE**: Dynamic runtime asset builds are not downloaded until they are needed for command execution.*

When Sensu finds a matching build, it downloads the build artifact from the specified URL. If the asset definition includes headers, they are passed along as part of the HTTP request. If the downloaded artifact's SHA512 checksum matches the checksum provided by the build, it is unpacked into the Sensu service's local cache directory.

Set the backend or agent's local cache path with the `cache-dir` configuration option. Disable dynamic runtime assets for an agent with the agent `disable-assets` configuration option.

> ***NOTE**: Dynamic runtime asset builds are unpacked into the cache directory that is configured with the `cache-dir` configuration option.*

Use the `assets-rate-limit` and `assets-burst-limit` configuration options for the agent and backend to configure a global rate limit for fetching dynamic runtime assets.

Dynamic runtime asset build execution

The directory path of each dynamic runtime asset listed in a check, event filter, handler, or mutator resource's `runtime_assets` array is appended to the `PATH` before the resource's `command` is executed. Subsequent check, event filter, handler, or mutator executions look for the dynamic runtime asset in the local cache and ensure that the contents match the configured checksum.

The following example demonstrates a use case with a Sensu check resource and an asset:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: sensu-prometheus-collector
spec:
  builds:
  - url:
https://assets.bonsai.sensu.io/ef812286f59de36a40e51178024b81c69666e1b7/sensu-
prometheus-collector_1.1.6_linux_amd64.tar.gz
    sha512:
a70056ca02662fbf2999460f6be93f174c7e09c5a8b12efc7cc42ce1ccb5570ee0f328a2dd8223f506df
3b5972f7f521728f7bdd6abf9f6ca2234d690aeb3808
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
---
type: CheckConfig
api_version: core/v2
metadata:
  name: prometheus_collector
spec:
  command: "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-query
up"
  interval: 10
  publish: true
  output_metric_handlers:
  - influxdb
  output_metric_format: influxdb_line
  runtime_assets:
  - sensu-prometheus-collector
  subscriptions:
  - system
```

JSON

```
{
  "type": "Asset",
```

```
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-email-handler"
  },
  "spec": {
    "builds": [
      {
        "url":
"https://assets.bonsai.sensu.io/45eaac0851501a19475a94016a4f8f9688a280f6/sensu-
email-handler_0.2.0_linux_amd64.tar.gz",
        "sha512":
"d69df76612b74acd64aef8eed2ae10d985f6073f9b014c8115b7896ed86786128c20249fd370f30672b
f9a11b041a99adb05e3a23342d3ad80d0c346ec23a946",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ]
      }
    ]
  }
}
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "prometheus_collector"
  },
  "spec": {
    "command": "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
query up",
    "handlers": [
    "influxdb"
    ],
    "interval": 10,
    "publish": true,
    "output_metric_format": "influxdb_line",
    "runtime_assets": [
      "sensu-prometheus-collector"
    ],
    "subscriptions": [
      "system"
    ]
```

```
    }
  }
```

Dynamic runtime asset format specification

Sensu expects a dynamic runtime asset to be a tar archive (optionally gzipped) that contains one or more executables within a bin folder. Any scripts or executables should be within a `bin/` folder in the archive. Read the Sensu Go Plugin template for an example dynamic runtime asset and Bonsai configuration.

The following are injected into the execution context:

- `{PATH_TO_ASSET}/bin` is injected into the `PATH` environment variable
- `{PATH_TO_ASSET}/lib` is injected into the `LD_LIBRARY_PATH` environment variable
- `{PATH_TO_ASSET}/include` is injected into the `CPATH` environment variable

***NOTE**: You cannot create a dynamic runtime asset by creating an archive of an existing project (as in previous versions of Sensu for plugins from the Sensu Plugins community). Follow the steps outlined in Contributing Assets for Existing Ruby Sensu Plugins, a Sensu Discourse guide. For further examples of Sensu users who have added the ability to use a community plugin as a dynamic runtime asset, read this Discourse post.*

Default cache directory

system	sensu-backend	sensu-agent
Linux	`/var/cache/sensu/sensu-backend`	`/var/cache/sensu/sensu-agent`
Windows	N/A	`C:\ProgramData\sensu\cache\sensu-agent`

If the requested dynamic runtime asset is not in the local cache, it is downloaded from the asset URL. The Sensu backend acts as an index of dynamic runtime asset builds, and does not provide storage or hosting for the build artifacts. Sensu expects dynamic runtime assets to be retrieved over HTTP or HTTPS.

Example dynamic runtime asset structure

```
sensu-example-handler_1.0.0_linux_amd64
├── CHANGELOG.md
├── LICENSE
├── README.md
└── bin
  └── my-check.sh
└── lib
└── include
```

Dynamic runtime asset path

When you download and install a dynamic runtime asset, the asset files are saved to a local path on disk. Most of the time, you won't need to know this path — except in cases where you need to provide the full path to dynamic runtime asset files as part of a command argument.

The dynamic runtime asset directory path includes the asset's checksum, which changes every time underlying asset artifacts are updated. This would normally require you to manually update the commands for any of your checks, handlers, hooks, or mutators that consume the dynamic runtime asset. However, because the dynamic runtime asset directory path is exposed to asset consumers via environment variables and the `assetPath` custom function, you can avoid these manual updates.

You can retrieve the dynamic runtime asset's path as an environment variable in the `command` context for checks, handlers, hooks, and mutators. Token substitution with the `assetPath` custom function is only available for check and hook commands.

The Sensu Windows agent uses `cmd.exe` for the check execution environment. For all other operating systems, the Sensu agent uses the Bourne shell (sh).

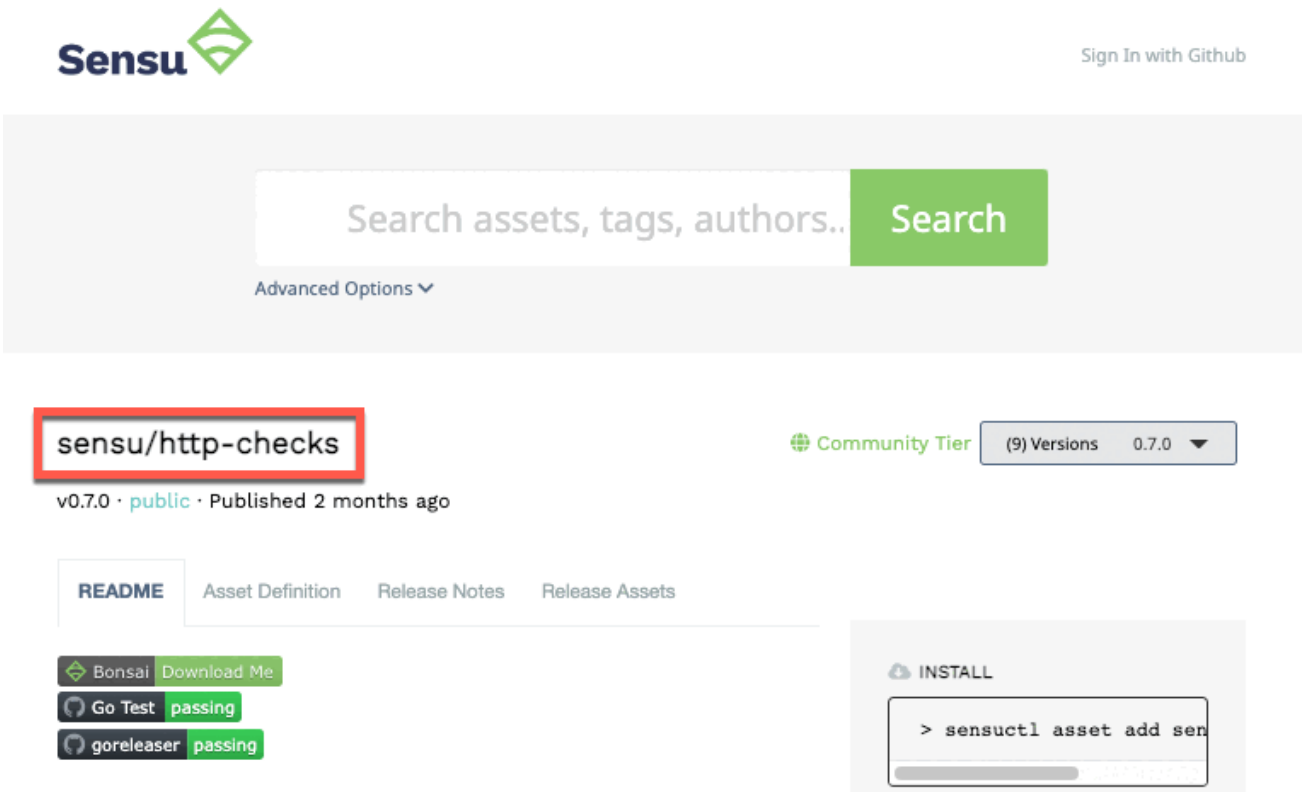
Environment variables for dynamic runtime asset paths

For each dynamic runtime asset, a corresponding environment variable will be available in the `command` context.

Sensu generates the environment variable name by capitalizing the dynamic runtime asset's complete

name, replacing any special characters with underscores, and appending the `_PATH` suffix. The value of the variable will be the path on disk where the dynamic runtime asset build has been unpacked.

Each asset page in Bonsai lists the asset's complete name. This example shows where the complete name for the sensu/http-checks dynamic runtime asset is located in Bonsai:



An asset's complete name includes both the part before the forward slash (sometimes called the Bonsai namespace) and the part after the forward slash.

Consequently, the environment variable for the sensu/http-checks asset path is:

```
SENSU_HTTP_CHECKS_PATH
```

Linux environment variable example

The Linux environment interprets the content between the `${` and `}` characters as an environment variable name and will substitute the value of that environment variable.

For example, to reference the path for the sensu/http-checks asset in your checks, handlers, hooks, and mutators:

```
${SENSU_HTTP_CHECKS_PATH}
```

Windows environment variable example

The Windows console environment interprets the content between paired `%` characters as an environment variable name and will substitute the value of that environment variable.

For example, to reference the path for the sensu/sensu-windows-powershell-checks asset in your checks, handlers, hooks, and mutators:

```
%SENSU_SENSU_WINDOWS_POWERSHELL_CHECKS_PATH%
```

assetPath function for dynamic runtime asset paths

The `assetPath` token subsitution function allows you to substitute a dynamic runtime asset's local path on disk so that you will not need to manually update your check or hook commands every time the asset is updated.

> ***NOTE**: The `assetPath` function is only available where token substitution is available: the `command` attribute of a check or hook resource. To access a dynamic runtime asset path in a handler or mutator command, you must use the environment variable.*

Linux assetPath example

To use the `assetPath` token substitution function in a Linux environment, place it immediately after the `$` character.

For example, to use the `assetPath` function to reference the path for the sensu/http-checks asset in your check or hook resources:

```
${{assetPath "sensu/http-checks"}}
```

Windows assetPath example

To use the `assetPath` token substitution function in a Linux environment, place it between paired `%` characters.

For example, to use the `assetPath` function to reference the path for the sensu/sensu-windows-powershell-checks asset in your check or hook resources:

```
%{{assetPath "sensu/sensu-windows-powershell-checks"}}%
```

When running PowerShell plugins on Windows, the exit status codes that Sensu captures may not match the expected values. To correctly capture exit status codes from PowerShell plugins distributed as dynamic runtime assets, use the asset path to construct the command. The following example uses the `assetPath` function for this purpose:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: win-cpu-check
spec:
  command: powershell.exe -ExecutionPolicy ByPass -f %{{assetPath "sensu/sensu-windows-powershell-checks"}}%\bin\check-windows-cpu-load.ps1 90 95
  subscriptions:
  - windows
  handlers:
  - slack
  - email
  runtime_assets:
  - sensu/sensu-windows-powershell-checks
  interval: 10
  publish: true
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
```

```
  "metadata": {
    "name": "win-cpu-check"
  },
  "spec": {
    "command": "powershell.exe -ExecutionPolicy ByPass -f %{{assetPath
\"sensu/sensu-windows-powershell-checks\"}}%\\bin\\check-windows-cpu-load.ps1 90
95",
    "subscriptions": [
      "windows"
    ],
    "handlers": [
      "slack",
      "email"
    ],
    "runtime_assets": [
      "sensu/sensu-windows-powershell-checks"
    ],
    "interval": 10,
    "publish": true
  }
}
```

Asset hello world Bourne shell example

In this example, you'll run a script that outputs `Hello World`:

hello-world.sh

```
#!/bin/sh

STRING="Hello World"

echo $STRING

if [ $? -eq 0 ]; then
    exit 0
else
    exit 2
fi
```

The first step is to ensure that your directory structure is in place. As noted in Example dynamic runtime asset structure, your script could live in three potential directories in the project: `/bin`, `/lib`, or `/include`. For this example, put your script in the `/bin` directory.

1. Create the directory `sensu-go-hello-world`:

   ```
   mkdir sensu-go-hello-world
   ```

2. Navigate to the `sensu-go-hello-world` directory:

   ```
   cd sensu-go-hello-world
   ```

3. Create the directory `/bin`:

   ```
   mkdir bin
   ```

4. Copy the script into the `/bin` directory:

   ```
   cp hello-world.sh bin/
   ```

5. Confirm that the script is in the `/bin` directory:

   ```
   tree
   ```

   The response should list the `hello-world.sh` script in the `/bin` directory:

   ```
   .
   └── bin
       └── hello-world.sh
   ```

If you receive a `command not found` response, install `tree` and run the command again.

6. Make sure that the script is marked as executable:

```
chmod +x bin/hello-world.sh
```

If you do not receive a response, the command was successful.

Now that the script is in the directory, move on to the next step: packaging the `sensu-go-hello-world` directory as a dynamic runtime asset tarball.

Package the dynamic runtime asset

Dynamic runtime assets are archives, so packaging the asset requires creating a tar.gz archive of your project.

1. Navigate to the directory you want to tar up.

2. Create the tar.gz archive:

```
tar -C sensu-go-hello-world -cvzf sensu-go-hello-world-0.0.1.tar.gz .
```

3. Generate a SHA512 sum for the tar.gz archive (this is required for the dynamic runtime asset to work):

```
sha512sum sensu-go-hello-world-0.0.1.tar.gz | tee sha512sum.txt
```

From here, you can host your dynamic runtime asset wherever you'd like. To make the asset available via Bonsai, you'll need to host it on GitHub. Learn more in The "Hello World" of Sensu Assets at the Sensu Community Forum on Discourse.

To host your dynamic runtime asset on a different platform like Gitlab or Bitbucket, upload your asset there. You can also use Artifactory or even Apache or NGINX to serve your asset. All that's required for your dynamic runtime asset to work is the URL to the asset and the SHA512 sum for the asset to be

downloaded.

Asset specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Dynamic runtime assets should always be type `Asset`.
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	**YML** `type: Asset` **JSON** `{ "type": "Asset" }`

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For dynamic runtime assets in this version of Sensu, the `api_version` should always be `core/v2`.
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

example

```
api_version: core/v2
```

JSON

```
{
  "api_version": "core/v2"
}
```

metadata

description

Top-level collection of metadata about the dynamic runtime asset, including `name`, `namespace`, and `created_by` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the asset definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.

required

Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.

type

Map of key-value pairs

YML

example

```
metadata:
  name: check_script
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    playbook: www.example.url
```

JSON

```
{
  "metadata": {
    "name": "check_script",
```

```
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "playbook": "www.example.url"
    }
  }
}
```

spec	
description	Top-level map that includes the dynamic runtime asset spec attributes.
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example (multiple builds)	YML

```
spec:
  builds:
  - url: http://example.com/asset-linux-amd64.tar.gz
    sha512:
487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a
72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fcc
b574c1a5d3
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    headers:
      Authorization: Bearer {{ .annotations.asset_token |
default "N/A" }}
      X-Forwarded-For: client1, proxy1, proxy2
  - url: http://example.com/asset-linux-armv7.tar.gz
    sha512:
70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d1538374066
a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f7
```

```
5670a8c68b
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'arm'
    - entity.system.arm_version == 7
    headers:
      Authorization: Bearer {{ .annotations.asset_token |
default "N/A" }}
      X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```
{
  "spec": {
    "builds": [
      {
        "url": "http://example.com/asset-linux-
amd64.tar.gz",
        "sha512":
"487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58
a72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fc
cb574c1a5d3",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": {
          "Authorization": "Bearer {{
.annotations.asset_token | default \"N/A\" }}",
          "X-Forwarded-For": "client1, proxy1, proxy2"
        }
      },
      {
        "url": "http://example.com/asset-linux-
armv7.tar.gz",
        "sha512":
"70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d153837406
6a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f
75670a8c68b",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'arm'",
```

```
          "entity.system.arm_version == 7"
        ],
        "headers": {
          "Authorization": "Bearer {{
.annotations.asset_token | default \"N/A\" }}",
          "X-Forwarded-For": "client1, proxy1, proxy2"
        }
      }
    ]
  }
}
```

YML

example (single build, deprecated)

```
spec:
  url: http://example.com/asset.tar.gz
  sha512:
4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3
ef3c2e8d154812246e5dda4a87450576b2c58ad9ab40c9e2edc31b288d0
66b195b21b
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
  headers:
    Authorization: Bearer {{ .annotations.asset_token |
default "N/A" }}
    X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```
{
  "spec": {
    "url": "http://example.com/asset.tar.gz",
    "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a
3ef3c2e8d154812246e5dda4a87450576b2c58ad9ab40c9e2edc31b288d
066b195b21b",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
```

```
    ],
    "headers": {
      "Authorization": "Bearer {{ .annotations.asset_token
| default \"N/A\" }}",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
}
```

Metadata attributes

name	
description	Unique name of the dynamic runtime asset, validated with Go regex `\A[\w\.\-]+\z`.
required	true
type	String
example	**YML** `name: check_script` **JSON** `{ "name": "check_script" }`

namespace	
description	Sensu RBAC namespace that the dynamic runtime asset belongs to.
required	false

type	String
default	`default`
example	**YML** ```yaml namespace: production ``` **JSON** ```json { "namespace": "production" } ```

created_by

description	Username of the Sensu user who created the dynamic runtime asset or last updated the asset. Sensu automatically populates the `created_by` field when the dynamic runtime asset is created or updated.
required	false
type	String
example	**YML** ```yaml created_by: admin ``` **JSON** ```json { "created_by": "admin" } ```

labels

description	Custom attributes to include with observation event data that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	**YML** `labels:` `  environment: development` `  region: us-west-2` **JSON** `{` `  "labels": {` `  "environment": "development",` `  "region": "us-west-2"` `  }` `}`

annotations

description	Non-identifying metadata to include with observation event data that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu.

	In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML** `annotations:` `  managed-by: ops` `  playbook: www.example.url` **JSON** `{` `  "annotations": {` `  "managed-by": "ops",` `  "playbook": "www.example.url"` `  }` `}`

Spec attributes

builds	
description	List of dynamic runtime asset builds used to define multiple artifacts that provide the named asset.
required	true, if `url`, `sha512` and `filters` are not provided
type	Array
example	**YML** `builds:` `- url: http://example.com/asset-linux-amd64.tar.gz` `  sha512:`

```
487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a
72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fcc
b574c1a5d3
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
- url: http://example.com/asset-linux-armv7.tar.gz
  sha512:
70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d1538374066
a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f7
5670a8c68b
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'arm'
  - entity.system.arm_version == 7
```

JSON

```
{
  "builds": [
    {
      "url": "http://example.com/asset-linux-amd64.tar.gz",
      "sha512":
"487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58
a72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fc
cb574c1a5d3",
      "filters": [
        "entity.system.os == 'linux'",
        "entity.system.arch == 'amd64'"
      ]
    },
    {
      "url": "http://example.com/asset-linux-armv7.tar.gz",
      "sha512":
"70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d153837406
6a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f
75670a8c68b",
      "filters": [
        "entity.system.os == 'linux'",
        "entity.system.arch == 'arm'",
        "entity.system.arm_version == 7"
      ]
```

```
    }
  ]
}
```

url

description	URL location of the dynamic runtime asset. You can use token substitution in the URLs of your asset definitions so each backend or agent can download dynamic runtime assets from the appropriate URL without duplicating your assets (for example, if you want to host your assets at different datacenters).
required	true, unless `builds` are provided
type	String

YML

```
url: http://example.com/asset.tar.gz
```

JSON

```
{
  "url": "http://example.com/asset.tar.gz"
}
```

sha512

description	Checksum of the dynamic runtime asset.
required	true, unless `builds` are provided
type	String

YML

```
sha512: 4f926bf4328...
```

JSON

```
{
  "sha512": "4f926bf4328..."
}
```

filters

description	Set of Sensu query expressions used to determine if the dynamic runtime asset should be installed. If multiple expressions are included, each expression must return `true` for Sensu to install the asset. Filters for *check* dynamic runtime assets should match agent entity platforms. Filters for *handler* and *filter* dynamic runtime assets should match your Sensu backend platform. You can create asset filter expressions using any supported entity.system attributes, including `os`, `arch`, `platform`, and `platform_family`. ***PRO TIP**: Dynamic runtime asset filters let you reuse checks across platforms safely. Assign dynamic runtime assets for multiple platforms to a single check, and rely on asset filters to ensure that only the appropriate asset is installed on each agent.*
required	false
type	Array
example	**YML** `filters:` `- entity.system.os=='linux'` `- entity.system.arch=='amd64'` **JSON** `{`

```
  "filters": [
    "entity.system.os=='linux'",
    "entity.system.arch=='amd64'"
  ]
}
```

headers

description	HTTP headers to apply to dynamic runtime asset retrieval requests. You can use headers to access secured dynamic runtime assets. For headers that require multiple values, separate the values with a comma. You can use token substitution in your dynamic runtime asset headers (for example, to include secure information for authentication).
required	false
type	Map of key-value string pairs
example	**YML** `headers:` `  Authorization: Bearer {{ .annotations.asset_token | default "N/A" }}` `  X-Forwarded-For: client1, proxy1, proxy2` **JSON** `{` `  "headers": {` `  "Authorization": "Bearer {{ .annotations.asset_token | default \"N/A\" }}",` `  "X-Forwarded-For": "client1, proxy1, proxy2"` `  }` `}`

Dynamic runtime asset filters based on entity.system attributes

Use the entity.system attributes in dynamic runtime asset filters to specify which systems and configurations an asset or asset builds can be used with.

For example, the sensu/sensu-ruby-runtime dynamic runtime asset definition includes several builds, each with filters for several `entity.system` attributes:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: sensu-ruby-runtime
  labels:
  annotations:
    io.sensu.bonsai.url: https://bonsai.sensu.io/assets/sensu/sensu-ruby-runtime
    io.sensu.bonsai.api_url: https://bonsai.sensu.io/api/v1/assets/sensu/sensu-ruby-runtime
    io.sensu.bonsai.tier: Community
    io.sensu.bonsai.version: 0.0.10
    io.sensu.bonsai.namespace: sensu
    io.sensu.bonsai.name: sensu-ruby-runtime
    io.sensu.bonsai.tags: ''
spec:
  builds:
  - url:
https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-runtime_0.0.10_ruby-2.4.4_centos6_linux_amd64.tar.gz
    sha512:
cbee19124b7007342ce37ff9dfd4a1dde03beb1e87e61ca2aef606a7ad3c9bd0bba4e53873c07afa5ac46b0861967a9224511b4504dadb1a5e8fb687e9495304
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    - entity.system.platform_family == 'rhel'
    - parseInt(entity.system.platform_version.split('.')[0]) == 6
  - url:
https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-runtime_0.0.10_ruby-2.4.4_debian_linux_amd64.tar.gz
```

```
    sha512:
a28952fd93fc63db1f8988c7bc40b0ad815eb9f35ef7317d6caf5d77ecfbfd824a9db54184400aa0c81c
29b34cb48c7e8c6e3f17891aaf84cafa3c134266a61a
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    - entity.system.platform_family == 'debian'
  - url:
https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
runtime_0.0.10_ruby-2.4.4_alpine_linux_amd64.tar.gz
    sha512:
8d768d1fba545898a8d09dca603457eb0018ec6829bc5f609a1ea51a2be0c4b2d13e1aa46139ecbb0487
3449e4c76f463f0bdfbaf2107caf37ab1c8db87d5250
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    - entity.system.platform == 'alpine'
    - entity.system.platform_version.split('.')[0] == '3'
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-ruby-runtime",
    "labels": null,
    "annotations": {
      "io.sensu.bonsai.url": "https://bonsai.sensu.io/assets/sensu/sensu-ruby-
runtime",
      "io.sensu.bonsai.api_url": "https://bonsai.sensu.io/api/v1/assets/sensu/sensu-
ruby-runtime",
      "io.sensu.bonsai.tier": "Community",
      "io.sensu.bonsai.version": "0.0.10",
      "io.sensu.bonsai.namespace": "sensu",
      "io.sensu.bonsai.name": "sensu-ruby-runtime",
      "io.sensu.bonsai.tags": ""
    }
  },
  "spec": {
    "builds": [
```

```
      {
        "url":
"https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
runtime_0.0.10_ruby-2.4.4_centos6_linux_amd64.tar.gz",
        "sha512":
"cbee19124b7007342ce37ff9dfd4a1dde03beb1e87e61ca2aef606a7ad3c9bd0bba4e53873c07afa5ac
46b0861967a9224511b4504dadb1a5e8fb687e9495304",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'",
          "entity.system.platform_family == 'rhel'",
          "parseInt(entity.system.platform_version.split('.')[0]) == 6"
        ]
      },
      {
        "url":
"https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
runtime_0.0.10_ruby-2.4.4_debian_linux_amd64.tar.gz",
        "sha512":
"a28952fd93fc63db1f8988c7bc40b0ad815eb9f35ef7317d6caf5d77ecfbfd824a9db54184400aa0c81
c29b34cb48c7e8c6e3f17891aaf84cafa3c134266a61a",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'",
          "entity.system.platform_family == 'debian'"
        ]
      },
      {
        "url":
"https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
runtime_0.0.10_ruby-2.4.4_alpine_linux_amd64.tar.gz",
        "sha512":
"8d768d1fba545898a8d09dca603457eb0018ec6829bc5f609a1ea51a2be0c4b2d13e1aa46139ecbb048
73449e4c76f463f0bdfbaf2107caf37ab1c8db87d5250",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'",
          "entity.system.platform == 'alpine'",
          "entity.system.platform_version.split('.')[0] == '3'"
        ]
      }
    ]
```

```
    }
  }
```

In this example, if you install the dynamic runtime asset on a system running Linux AMD64 Alpine version 3.xx, Sensu will ignore the first two builds and install the third.

> ***NOTE**: Sensu downloads and installs the first build whose filter expressions all evaluate as `true`. If your system happens to match all of the filters for more than one build of a dynamic runtime asset, Sensu will only install the first build.*

All of the dynamic runtime asset filter expressions must evaluate as true for Sensu to download and install the asset and run the check, handler, or filter that references the asset.

To continue this example, if you try to install the dynamic runtime asset on a system running Linux AMD64 Alpine version 2.xx, the `entity.system.platform_version` argument will evaluate as `false`. In this case, the asset will not be downloaded and the check, handler, or filter that references the asset will fail to run.

Add dynamic runtime asset filters to specify that an asset is compiled for any of the entity.system attributes, including operating system, platform, platform version, and architecture. Then, you can rely on dynamic runtime asset filters to ensure that you install only the appropriate asset for each of your agents.

Share an asset on Bonsai

Share your open-source dynamic runtime assets on Bonsai and connect with the Sensu community. Bonsai supports dynamic runtime assets hosted on GitHub and released using GitHub releases. For more information about creating Sensu plugins, read the plugins reference.

Bonsai requires a `bonsai.yml` configuration file in the root directory of your repository that includes the project description, platforms, asset filenames, and SHA-512 checksums. For a Bonsai-compatible dynamic runtime asset template using Go and GoReleaser, review the Sensu Go plugin skeleton.

To share your dynamic runtime asset on Bonsai, log in to Bonsai with your GitHub account and authorize Sensu. After you are logged in, you can register your dynamic runtime asset on Bonsai by adding the GitHub repository, a description, and tags. Make sure to provide a helpful README for your dynamic runtime asset with configuration examples.

`bonsai.yml` example

```
---
description: "#{repo}"
builds:
- platform: "linux"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'linux'"
  -  "entity.system.arch == 'amd64'"

- platform: "Windows"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_windows_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'windows'"
  -  "entity.system.arch == 'amd64'"
```

`bonsai.yml` specification

description

description	Project description.
required	true
type	String
example	`description: "#{repo}"`

builds

description	Array of dynamic runtime asset details per platform.
required	true
type	Array
example	

```
builds:
- platform: "linux"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'linux'"
  -  "entity.system.arch == 'amd64'"
```

Builds specification

platform

description	Platform supported by the dynamic runtime asset.
required	true
type	String
example	

```
- platform: "linux"
```

arch

description	Architecture supported by the dynamic runtime asset.
required	true
type	String
example	

```
arch: "amd64"
```

asset_filename

description	File name of the archive that contains the dynamic runtime asset.
required	true
type	String
example	`asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"`

sha_filename

description	SHA-512 checksum for the dynamic runtime asset archive.
required	true
type	String
example	`sha_filename: "#{repo}_#{version}_sha512-checksums.txt"`

filter

description	Filter expressions that describe the operating system and architecture supported by the asset.
required	false
type	Array
example	

```
filter:
-  "entity.system.os == 'linux'"
```

```
    - "entity.system.arch == 'amd64'"
```

Delete dynamic runtime assets

Delete dynamic runtime assets with a DELETE request to the `/assets` API endpoint or with the `sensuctl asset delete` command.

Removing a dynamic runtime asset from Sensu *does not* remove references to the deleted asset in any other resource (including checks, filters, mutators, handlers, and hooks). You must also update resources and remove any reference to the deleted dynamic runtime asset. Failure to do so will result in errors like `sh: asset.sh: command not found`.

Errors as a result of failing to remove the dynamic runtime asset from checks and hooks will surface in the event data. Errors as a result of failing to remove the dynamic runtime asset reference on a mutator, handler, or filter will only surface in the backend logs.

Deleting a dynamic runtime asset does not delete the archive or downloaded files on disk. You must remove the archive and downloaded files from the asset cache manually.

Plugins reference

Sensu plugins provide executable scripts or other programs that you can use as Sensu checks, handlers, and mutators. Sensu plugins must comply with the following specification:

- Accept input/data via stdin (handler and mutator plugins only)
  - Optionally able to parse a JSON data payload (that is, observation data in an event)
- Output data to stdout or stderr
- Produce an exit status code to indicate state:
  - `0` indicates `OK`
  - `1` indicates `WARNING`
  - `2` indicates `CRITICAL`
  - exit status codes other than `0`, `1`, or `2` indicate an unknown or custom status
- Optionally able to parse command line arguments to modify plugin behavior

Supported programming languages

You can use any programming language that can satisfy the Sensu plugin specification requirements — which is nearly any programming language in the world — to write Sensu plugins.

Using plugins written in programming languages other than Go requires you to install the corresponding runtime. For example, to use a Ruby plugin with Sensu Go, you must install the Sensu Go Ruby Runtime asset.

Use Nagios plugins

The Sensu plugin specification is compatible with the Nagios plugin specification, so you can use the 50+ plugins in the official Nagios Plugins project and 4000+ plugins in the Nagios Exchange with Sensu without any modification.

Plugin execution

All plugins are executed by the Sensu backend. Plugins must be executable files that are discoverable on the Sensu system (that is, installed in a system `$PATH` directory) or referenced with an absolute path (for example, `/opt/path/to/my/plugin`).

***NOTE**: By default, Sensu installer packages will modify the system `$PATH` for the Sensu processes to include `/etc/sensu/plugins`. As a result, executable scripts (for example, plugins) located in `/etc/sensu/plugins` will be valid commands. This allows command attributes to use relative paths for Sensu plugin commands, such as `"command": "http-check --url https://sensu.io"`.*

Plugin configuration overrides

Many plugins support configuration overrides on a per-entity or per-check basis. For example, some plugins allow you to use annotations in individual entities and checks to set arguments that will override any arguments set in a resource command or in backend runtime environment variables for only that entity or check.

Read the Bonsai documentation for a plugin to learn about any configuration overrides the plugin supports.

Go plugin example

The following example shows the structure for a very basic Sensu Go plugin.

```
package main

import (
    "fmt"
    "log"

    "github.com/sensu-community/sensu-plugin-sdk/sensu"
    "github.com/sensu/sensu-go/types"
)
```

```
// Config represents the check plugin config.
type Config struct {
    sensu.PluginConfig
    Example string
}

var (
    plugin = Config{
  PluginConfig: sensu.PluginConfig{
   Name:     "check_name",
   Short:    "Description for check_name",
   Keyspace: "sensu.io/plugins/check_name/config",
  },
    }

    options = []*sensu.PluginConfigOption{
  &sensu.PluginConfigOption{
   Path:      "example",
   Env:       "CHECK_EXAMPLE",
   Argument:  "example",
   Shorthand: "e",
   Default:   "",
   Usage:     "An example string configuration option",
   Value:     &plugin.Example,
  },
    }
)

func main() {
    check := sensu.NewGoCheck(&plugin.PluginConfig, options, checkArgs, executeCheck,
false)
    check.Execute()
}

func checkArgs(event *types.Event) (int, error) {
    if len(plugin.Example) == 0 {
  return sensu.CheckStateWarning, fmt.Errorf("--example or CHECK_EXAMPLE
environment variable is required")
    }
    return sensu.CheckStateOK, nil
}
```

```
func executeCheck(event *types.Event) (int, error) {
    log.Println("executing check with --example", plugin.Example)
    return sensu.CheckStateOK, nil
}
```

To create this scaffolding for a Sensu Go check, handler, mutator, or sensuctl plugin, use the Sensu Plugin Tool along with a default plugin template. The plugin template repositories wrap the Sensu Plugin SDK, which provides the framework for building Sensu Go plugins.

For a step-by-step walkthrough, read How to publish an asset with the Sensu Go SDK — you'll learn how to create a check plugin and a handler plugin with the Sensu Plugin SDK. You can also watch our 30-minute webinar, Intro to assets with the Sensu Go SDK, and learn to build a check plugin for Sensu Go.

Ruby plugin example

The following example demonstrates a very basic Sensu plugin in the Ruby programming language.

```
#!/usr/bin/env ruby
#
require 'json'

# Read the incoming JSON data from stdin
event = JSON.parse(stdin.read, :symbolize_names => true)

# Create an output object using Ruby string interpolation
output = "The check named #{event[:check][:name]} generated the following output: #
{event[:output]}"

# Convert the mutated event data back to JSON and output it to stdout
puts output
```

***NOTE**: This example is intended as a starting point for building a basic custom plugin in Ruby. It does not provide functionality.*

Install Sensu plugins

Extend Sensu's functionality with plugins, which provide executables for performing status or metric checks, mutators for changing data to a desired format, and handlers for performing an action on a Sensu event.

Install plugins with dynamic runtime assets

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. Read Use dynamic runtime assets to install plugins to become familiar with workflows that involve assets.

> ***NOTE**: Dynamic runtime assets are not required to use Sensu Go. You can install Sensu plugins using the sensu-install tool or a configuration management solution.*

Use the Sensu Catalog to find, configure, and install many plugins directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.

You can also use Bonsai, the Sensu asset hub, a centralized place for downloading and sharing dynamic runtime assets. Bonsai lists hundreds of plugins, libraries, and runtimes with instructions and examples to help you automate your monitoring workflows. You can also share your asset on Bonsai.

Install plugins with the sensu-install tool

To use community plugins that are not yet compatible with Sensu Go, use the `sensu-install` tool.

If you've used previous versions of Sensu, you're probably familiar with the Sensu Community Plugins organization on GitHub. Although some of these plugins are enabled for Sensu Go, some do not include the components necessary to work with Sensu Go. Read each plugin's instructions for information about whether it is compatibile with Sensu Go.

> ***NOTE**: Plugins in the Sensu Plugins GitHub organization are community-maintained: anyone can*

improve on them. To get started with adding to a plugin or sharing your own, head to the Sensu Community Slack channel. Maintainers are always happy to help answer questions and point you in the right direction.

The `sensu-install` tool comes with an embedded version of Ruby, so you don't need to have Ruby installed on your system.

To install a Sensu Community plugin with Sensu Go:

1. Install the sensu-plugins-ruby package from packagecloud.

2. Run the `sensu-install` command to install plugins in the Sensu Community Plugins GitHub organization by repository name. Plugins are installed into `/opt/sensu-plugins-ruby/embedded/bin`.

To list all flags for the sensu-install command, run:

```
sensu-install --help
```

The response will be similar to this example:

```
Usage: sensu-install [options]
    -h, --help                       Display this message
    -v, --verbose                    Enable verbose logging
    -p, --plugin PLUGIN              Install a Sensu PLUGIN
    -P, --plugins PLUGIN[,PLUGIN]    PLUGIN or comma-delimited list of Sensu plugins
to install
    -e, --extension EXTENSION        Install a Sensu EXTENSION
    -E, --extensions EXTENSION[,EXT] EXTENSION or comma-delimited list of Sensu
extensions to install
    -s, --source SOURCE              Install Sensu plugins and extensions from a
custom SOURCE
    -c, --clean                      Clean up (remove) other installed versions of
the plugin(s) and/or extension(s)
    -x, --proxy PROXY                Install Sensu plugins and extensions via a
PROXY URL
```

For example, to install the Sensu InfluxDB plugin:

```
sudo sensu-install -p influxdb
```

To install a specific version of the Sensu InfluxDB plugin with `sensu-install`, run:

```
sudo sensu-install -p 'sensu-plugins-influxdb:2.0.0'
```

***NOTE**: We recommend specifying the plugin version you want to install to maintain the stability of your observability infrastructure. If you do not specify a version to install, Sensu automatically installs the latest version, which may include breaking changes.*

Use a configuration management tool or Sensu dynamic runtime assets to pin the versions of any plugins installed in production.

***NOTE**: If a plugin is not Sensu Go-enabled and there is no analogue on Bonsai, you can add the necessary functionality to make the plugin compatible with Sensu Go. Follow the Discourse guide Contributing Assets for Existing Ruby Sensu Plugins to walk through the process.*

Troubleshoot the sensu-install tool

Some plugins require additional tools to install them successfully. An example is the Sensu disk checks plugin.

To download and update package information:

SHELL

```
sudo apt-get update
```

SHELL

```
sudo yum update
```

Depending on the plugin, you may need to install developer tool packages:

SHELL

```
sudo apt-get install build-essential
```

SHELL

```
sudo yum groupinstall "Development Tools"
```

Use dynamic runtime assets to install plugins

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. You can use assets to provide the plugins, libraries, and runtimes you need to automate your monitoring workflows. Read the asset reference for more information about dynamic runtime assets. This guide uses the sensu/sensu-pagerduty-handler dynamic runtime asset as an example.

***NOTE**: Dynamic runtime assets are not required to use Sensu Go. You can install Sensu plugins using the sensu-install tool or a configuration management solution.*

Register an asset

To add the sensu/sensu-pagerduty-handler dynamic runtime asset to Sensu, use sensuctl asset add:

```
sensuctl asset add sensu/sensu-pagerduty-handler:2.2.0 -r pagerduty-handler
```

The response should be similar to this example:

```
fetching bonsai asset: sensu/sensu-pagerduty-handler:2.2.0
added asset: sensu/sensu-pagerduty-handler:2.2.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["pagerduty-handler"].
```

***NOTE**: Specify the asset version you want to install to maintain the stability of your observability infrastructure. If you do not specify a version to install, Sensu automatically installs the latest*

version, which may include breaking changes.

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `pagerduty-handler`.

You can also open the **Release Assets** tab on asset pages in Bonsai to download the asset definition for your Sensu backend platform and architecture.

***NOTE**: Sensu does not download and install asset builds onto the system until they are needed for command execution. Read the asset reference for more information about asset builds.*

If you are using a Sensu package, the asset is installed at `/var/cache`. If you are using a Sensu Docker image, the asset is installed at `/var/lib`.

Adjust the asset definition

Asset definitions tell Sensu how to download and verify the asset when required by a check, filter, mutator, or handler.

After you add or download the asset definition, open the file and adjust the `namespace` and `filters` for your Sensu instance. Here's the asset definition for version 2.2.0 of the sensu/sensu-pagerduty-handler asset for Linux AMD64:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  annotations:
    io.sensu.bonsai.api_url: https://bonsai.sensu.io/api/v1/assets/sensu/sensu-pagerduty-handler
    io.sensu.bonsai.name: sensu-pagerduty-handler
    io.sensu.bonsai.namespace: sensu
    io.sensu.bonsai.tags: handler
    io.sensu.bonsai.tier: Supported
    io.sensu.bonsai.url: https://bonsai.sensu.io/assets/sensu/sensu-pagerduty-handler
    io.sensu.bonsai.version: 2.2.0
```

```
  name: pagerduty-handler
spec:
  builds:
  - filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    headers: null
    sha512:
adc6ee846b88a792cc0f384a942f8b7ff727c7d7cf6a3012a0bf97ae4bef770503f9d5c26f756047559c
145ac01c62d4db9af8574d0cc451a176f1be29f52ffc
    url:
https://assets.bonsai.sensu.io/87f00332d6f36f59ee188e9e2a94a2b84172d134/sensu-
pagerduty-handler_2.2.0_linux_amd64.tar.gz
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "annotations": {
      "io.sensu.bonsai.api_url": "https://bonsai.sensu.io/api/v1/assets/sensu/sensu-
pagerduty-handler",
      "io.sensu.bonsai.name": "sensu-pagerduty-handler",
      "io.sensu.bonsai.namespace": "sensu",
      "io.sensu.bonsai.tags": "handler",
      "io.sensu.bonsai.tier": "Supported",
      "io.sensu.bonsai.url": "https://bonsai.sensu.io/assets/sensu/sensu-pagerduty-
handler",
      "io.sensu.bonsai.version": "2.2.0"
    },
    "name": "pagerduty-handler"
  },
  "spec": {
    "builds": [
      {
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": null,
        "sha512":
```

```
"adc6ee846b88a792cc0f384a942f8b7ff727c7d7cf6a3012a0bf97ae4bef770503f9d5c26f756047559
c145ac01c62d4db9af8574d0cc451a176f1be29f52ffc",
        "url":
"https://assets.bonsai.sensu.io/87f00332d6f36f59ee188e9e2a94a2b84172d134/sensu-
pagerduty-handler_2.2.0_linux_amd64.tar.gz"
      }
    ]
  }
}
```

Filters for *check* dynamic runtime assets should match entity platforms. Filters for *handler and filter* dynamic runtime assets should match your Sensu backend platform. If the provided filters are too restrictive for your platform, replace `os` and `arch` with any supported entity system attributes (for example, `entity.system.platform_family == 'rhel'`). You may also want to customize the asset `name` to reflect the supported platform (for example, `pagerduty-handler-linux`) and add custom attributes with `labels` and `annotations`.

Enterprise-tier dynamic runtime assets (like the ServiceNow and Jira event handlers) require a Sensu commercial license. For more information about commercial features and to activate your license, read Get started with commercial features.

Use sensuctl to verify that the asset is registered and ready to use:

SHELL

```
sensuctl asset list --format yaml
```

SHELL

```
sensuctl asset list --format wrapped-json
```

Create a workflow

With the asset downloaded and registered, you can use it in a monitoring workflow. Dynamic runtime assets may provide executable plugins intended for use with a Sensu check, handler, mutator, or hook, or JavaScript libraries intended to provide functionality for use in event filters. The details in Bonsai are the best resource for information about each asset's capabilities and configuration.

For example, to use the Sensu PagerDuty Handler asset, you would create a `pagerduty` handler that includes your PagerDuty service API key in place of `SECRET` and `pagerduty-handler` as a runtime asset:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  name: pagerduty
spec:
  command: sensu-pagerduty-handler
  env_vars:
  - PAGERDUTY_TOKEN=SECRET
  filters:
  - is_incident
  runtime_assets:
  - pagerduty-handler
  timeout: 10
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "pagerduty"
  },
  "spec": {
    "type": "pipe",
    "command": "sensu-pagerduty-handler",
    "env_vars": [
      "PAGERDUTY_TOKEN=SECRET"
    ],
    "runtime_assets": [
      "pagerduty-handler"
    ],
    "timeout": 10,
    "filters": [
      "is_incident"
```

```
      ]
    }
  }
```

Save the definition to a file (for example, `pagerduty-handler.yml` or `pagerduty-handler.json`), and add it to Sensu with sensuctl:

SHELL

```
sensuctl create --file pagerduty-handler.yml
```

SHELL

```
sensuctl create --file pagerduty-handler.json
```

Now that Sensu can create incidents in PagerDuty, you can automate this workflow by adding the `pagerduty` handler to your Sensu service check definitions.Read Monitor server resources to learn more.

Next steps

Read these resources for more information about using dynamic runtime assets in Sensu:

- Assets reference
- Asset format specification
- Share assets on Bonsai

Follow Send PagerDuty alerts with Sensu to configure a check that generates status events and a handler that sends Sensu alerts to PagerDuty for non-OK events.

Featured Integrations

Sensu integrations include plugins, libraries, and runtimes that extend Sensu's functionality and allow you to automate your monitoring and observability workflows. You can also rely on Sensu's integrations to get work done with Sensu as part of your existing workflows.

Integrations are service-specific and have different setup and configuration requirements. Each integration has self-contained documentation with in-depth information about how to install and use it. Many of the featured integrations include curated quick-start templates that you only need to edit to match your configuration.

Although this category focuses on our most popular featured integrations, you can find more supported-, Enterprise-, and community-tier integrations at Bonsai, the Sensu asset hub.

Alerting and incident management

- Email
- Jira
- PagerDuty
- ServiceNow
- Slack

Auto-remediation

- Ansible
- Rundeck
- SaltStack

Deregistration

- Chef
-

- EC2
- Puppet

Time-series and long-term event storage

- Elasticsearch
- Graphite
- InfluxDB
- OpenTSDB
- Prometheus
- Sumo Logic
- TimescaleDB
- Wavefront

Ansible integration

***COMMERCIAL FEATURE**: Access the Sensu Ansible Handler integration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu Ansible Handler plugin is a Sensu handler that launches Ansible Tower job templates for automated remediation based on Sensu observability event data.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

The Sensu Ansible Handler plugin supports both Ansible Tower and Ansible AWX implementations of the Ansible Tower API, authenticating using Ansible Tower API tokens.

- Specify a default Ansible Tower job template for remediation actions for all checks and use check annotations to override the default as needed on a check-by check-basis.
- Automatically limit Ansible jobs to the host that matches the Sensu entity name encoded in a Sensu event.
- Optional job template requests: use Sensu check annotations to specify a set of Ansible Tower job template requests to run for matching Sensu event occurrence and severity conditions.
- Keep your Ansible Tower host and auth token secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Ansible Handler plugin, use the Sensu Catalog in the web UI to configure and install it. Or, use our curated, configurable quick-start template to integrate Sensu with your existing Ansible Tower workflows.

You can also add the Sensu Ansible Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Ansible workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Configuration management

Use the official Sensu Go Ansible Collection for configuration management for your Sensu instance. The documentation site includes installation instructions, example playbooks, and module references

Chef integration

The Sensu Chef Handler plugin is a Sensu handler that deletes a Sensu entity with a failing keepalive check when the entity's corresponding Chef node no longer exists.

> ***NOTE**: The Sensu Chef Handler plugin is an example of Sensu's deregistration integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Use Sensu annotations to override Sensu entity names with corresponding Chef node names.
- Keep your sensitive API authentication information secure with Sensu environment variables and secrets management.

Get the plugin

Add the Sensu Chef Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Chef workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Configuration management

Use the official Chef Cookbook for Sensu Go for configuration management for your Sensu instance.

EC2 integration

The Sensu EC2 Handler plugin is a Sensu handler that checks an AWS EC2 instance and removes it from Sensu if it is not in one of the specified states.

***NOTE**: The Sensu EC2 Handler plugin is an example of Sensu's deregistration integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Tunable arguments: use Sensu annotations to set custom instance ID, instance ID labels, timeouts, and more in EC2.
- Specify custom values for Sensu event metric points via metric tags.
- Keep your AWS EC2 API token, username, and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu EC2 Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing AWS EC2 workflows.

You can also add the Sensu EC2 Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing EC2 workflows.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

Set up a limited service account with the access and permissions required to automatically remove AWS EC2 instances using the Sensu EC2 Handler integration.

Elasticsearch integration

***COMMERCIAL FEATURE**: Access the Sensu Elasticsearch Handler integration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu Elasticsearch Handler plugin is a Sensu handler that sends observation data from Sensu events and metrics to Elasticsearch. With this handler, the Sensu observation data you send to Elasticsearch is available for indexing and visualization in Kibana.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Query metrics points within Elasticsearch: the handler automatically mutates metrics data by creating a top-level object with metric point names and their associated values.
- Index entire events for searching within Kibana.
- Use daily, weekly, monthly, and yearly index specification (for example, sensu_events-2020-11-10).
- Omit the transmission of certain redundant event fields to reduce the number of items indexed.
- Specify custom values for Sensu event metric points via metric tags.
- Use event-based templating to include observation data from event attributes to add meaningful, actionable context.
- Keep your Elasticsearch username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Elasticsearch Handler plugin, use the Sensu Catalog in the

web UI to configure and install it.Or, use our curated, configurable quick-start template for events and metrics data storage.

You can also add the Sensu Elasticsearch Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Elasticsearch workflows.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Email integration

The Sensu Email Handler plugin is a Sensu handler that sends email alerts based on your event data. With this handler, Sensu can send email messages to the email addresses you specify based on event data generated by your Sensu checks.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

The Sensu Email Handler plugin supports the login authentication mechanisms required for use with Google Mail, Office 365, and other standards-based email providers and transports.

- Use event-based templating to include observation data from event attributes to add meaningful, actionable context to your email alert messages.
- Keep your email provider username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Email Handler plugin, use the Sensu Catalog in the web UI to configure and install it.Or, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and send email alerts.

You can also add the Sensu Email Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing email workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

Walk through adding and configuring the Sensu Email Handler asset in the Send email alerts with the Sensu Go Email Handler guide.

Graphite integration

The Sensu Graphite Handler plugin is a Sensu handler that sends Sensu metrics to the time-series database Graphite so you can store, instrument, and visualize Sensu metrics data.

***NOTE**: The Sensu Graphite Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Transform metrics to Graphite format: extract and transform the metrics you collect from different sources in formats like Influx, Nagios, and OpenTSDB and populate them into Graphite.
- Specify custom values for Sensu event metric points via metric tags.
- Keep your Graphite host and port secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Graphite Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in Graphite.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Graphite Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

InfluxDB integration

The Sensu InfluxDB Handler plugin is a Sensu handler that sends Sensu metrics to the time-series database InfluxDB so you can store, instrument, and visualize Sensu metrics data. You can also use the Sensu InfluxDB Handler integration to create metrics from Sensu status check results for long-term storage in InfluxDB.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Transform metrics to InfluxDB format: extract and transform the metrics you collect from different sources in formats like Graphite, OpenTSDB, Nagios, and Influx and populate them into InfluxDB.
- Mutate check status into metrics to be stored in InfluxDB.
- Specify custom values for Sensu event metric points via metric tags.
- Keep your InfluxDB username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu InfluxDB Handler plugin, use the Sensu Catalog in the web UI to configure and install it. Or, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in InfluxDB.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu InfluxDB Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Jira integration

***COMMERCIAL FEATURE**: Access the Sensu Jira Handler integration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu Jira Handler plugin is a Sensu handler that creates and updates Jira issues based on observation data from Sensu events.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Tunable arguments: use Sensu annotations to set custom project names, issue types, resolution states, and more in Jira
- Use event-based templating to include observation data from event attributes to add meaningful, actionable context.
- Keep your Jira username, password, and API token secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Jira Handler plugin, use our curated, configurable quick-start template to send alerts based on Sensu events to Jira Service Desk.

Add the Sensu Jira Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Jira workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

OpenTSDB integration

The Sensu OpenTSDB Handler plugin is a Sensu handler that sends Sensu metrics to an OpenTSDB server via its Telnet-style API. This allows you to extract, tag, and store Sensu metrics data in an OpenTSDB database.

> ***NOTE**: The Sensu OpenTSDB Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Transform metrics to OpenTSDB format: extract and transform the metrics you collect from different sources in formats like Graphite, Influx, and Nagios and populate them into OpenTSDB.
- Specify custom values for Sensu event metric points via metric tags.

Get the plugin

To build your own workflow or integrate Sensu with existing workflows, add the Sensu OpenTSDB Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

PagerDuty integration

The Sensu PagerDuty Handler plugin is a Sensu handler that manages PagerDuty incidents and operator alerts. With this handler, Sensu can trigger and resolve PagerDuty incidents according to the PagerDuty schedules, notifications, and escalation, response, and orchestration workflows you already have configured.

> ***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Optional severity mapping: match Sensu check statuses with PagerDuty incident severities via a JSON document.
- Use event-based templating to create deduplication key arguments to group repeated alerts into one incident and summary template arguments to make sure your PagerDuty notifications include the event data your operators need to take action.
- Authenticate and route alerts based on PagerDuty teams using check and agent annotations.
- Keep your PagerDuty integration key secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu PagerDuty Handler plugin, use the Sensu Catalog in the web UI to configure and install it. Or, use our curated, configurable quick-start template for incident management to integrate Sensu with your existing PagerDuty workflows.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu PagerDuty Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

- Follow the Use dynamic runtime assets to install plugins guide to learn how to add and configure Sensu PagerDuty Handler asset.
- Demo the Sensu PagerDuty Handler integration with the Send Sensu Go alerts to PagerDuty guide.

Prometheus integrations

Sensu has two Prometheus plugins: the Prometheus Collector and the Prometheus Pushgateway Handler. Both help you get Sensu observability data into Prometheus.

> ***NOTE**: The Sensu Prometheus plugins are examples of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Sensu Prometheus Collector

The Sensu Prometheus Collector plugin is a Sensu check that collects metrics from a Prometheus exporter or the Prometheus query API and outputs the metrics to stdout in Influx, Graphite, or JSON format.

Features

- Turn Sensu into a super-powered Prometheus metric poller with Sensu's publish/subscribe model and client auto-registration (discovery) capabilities.
- Configure your Sensu instance to deliver the collected metrics to a time-series database like InfluxDB or Graphite.
- Specify custom values for Sensu event metric points via metric tags.
- Keep metrics endpoint authentication information secure with Sensu environment variables and secrets management.

Sensu Prometheus Pushgateway Handler

The Sensu Prometheus Pushgateway Handler plugin is a Sensu handler that sends Sensu metrics to a Prometheus Pushgateway, which Prometheus can then scrape.

Features

-

- Collect metrics by several means, including 20-year-old Nagios plugins with perfdata, and store them in Prometheus.
- Use default Prometheus metric type, job name, and instance name or specify custom values for Sensu event metric points via metric tags.

Get the plugins

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Prometheus plugins with a dynamic runtime asset from Bonsai, the Sensu asset hub.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

- Sensu Prometheus Collector plugin
- Sensu Prometheus Pushgateway Handler

Puppet integration

The Sensu Puppet Keepalive Handler plugin is a Sensu handler that deletes a Sensu entity with a failing keepalive check when the entity's corresponding Puppet node no longer exists or is deregistered.

***NOTE**: The Sensu Puppet Keepalive Handler plugin is an example of Sensu's deregistration integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Use Sensu annotations to override Sensu entity names with corresponding Puppet node names.
- Keep sensitive API authentication information secure with Sensu environment variables and secrets management.

Get the plugin

Add the Sensu Puppet Keepalive Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Puppet workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Configuration management

Use the partner-supported Sensu Puppet module for configuration management for your Sensu instance.

Rundeck integration

***COMMERCIAL FEATURE**: Access the Sensu Rundeck Handler integration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu Rundeck Handler plugin is a Sensu handler that initiates Rundeck jobs for automated remediation based on Sensu event data.

***NOTE**: The Sensu Rundeck Handler plugin is an example of Sensu's auto-remediation integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

The Sensu Rundeck Handler plugin supports both Rundeck Enterprise and Rundeck Open Source and standard job invocation or webhook invocation.

- Specify Rundeck jobs and webhooks along with trigger parameters for remediation actions for a check with Sensu annotations.
- Use event-based templating to make use of event data to specify the node to target for rememdiation.
- Keep your Rundeck auth token and webhook secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Rundeck Handler plugin, use one of our curated, configurable quick-start templates:

- Rundeck job
- Rundeck webhook

You can also add the Sensu Rundeck Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Rundeck workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

SaltStack integration

***COMMERCIAL FEATURE**: Access the Sensu SaltStack Handler integration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu SaltStack Handler plugin is a Sensu handler that launches SaltStack functions for automated remediation based on Sensu event data.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

The Sensu SaltStack Handler plugin supports both SaltStack Enterprise and SaltStack Open Source as well as SaltStack functions such as `service`, `state`, `saltutil`, and `grains` (including `arg` and `kwarg` arguments).

- Specify SaltStack functions and trigger parameters for remediation actions for a check with Sensu annotations.
- Use event-based templating to specify the minion to target for rememdiation based on event data.
- Keep your SaltStack username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu SaltStack Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing SaltStack workflows.

You can also add the Sensu SaltStack Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing SaltStack workflows.

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

ServiceNow integration

***COMMERCIAL FEATURE**: Access the Sensu ServiceNow Handler integration in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

The Sensu ServiceNow Handler plugin is a Sensu handler that creates and updates ServiceNow incidents and events based on observation data from Sensu events.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Automatically create a ServiceNow Configuration Item if none currently exists for a particular Sensu entity.
- Tunable arguments: use Sensu annotations to set custom incident notes, event information, Configuration Item descriptions, and more in ServiceNow.
- Use event-based templating to include observation data from event attributes to add meaningful, actionable context to ServiceNow incidents, events, and Configuration Items.
- Keep your ServiceNow username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu ServiceNow Handler plugin, use one of our curated, configurable quick-start templates:

- ServiceNow Incident Management: send Sensu observability alerts to ServiceNow Incident Management.
- ServiceNow Event Management: send Sensu observability data to ServiceNow Event

Management.

- ServiceNow Configuration Management Database (CMDB): register Sensu entities as configuration items in ServiceNow CMDB.

You can also add the Sensu ServiceNow Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing ServiceNow workflows.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Slack integration

The Sensu Slack Handler plugin is a Sensu handler that sends alerts based on your event data. With this handler, Sensu can trigger alerts to the Slack channels you specify based on event data generated by your Sensu checks.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Use event-based templating to include observation data from event attributes in your alerts to add meaningful, actionable context.
- Keep your Slack webhook secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Slack Handler plugin, use the Sensu Catalog in the web UI to configure and install it. Or, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and send alerts to Slack channels.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Slack Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

Read Send Slack alerts with handlers to learn how to add and configure the Sensu Slack Handler plugin.

Sumo Logic integration

The Sensu Sumo Logic Handler plugin is a Sensu handler that sends Sensu observability events and metrics to a Sumo Logic HTTP Logs and Metrics Source. This handler sends Sensu events as log entries, a set of metrics, or both, depending on the mode of operation you specify.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Query events and metrics points within Sumo Logic: the handler automatically mutates metrics data by creating a top-level object with metric point names and their associated values.
- Tunable arguments: use Sensu annotations to set Sumo Logic source name, host, and category; metric dimensions; log fields; and more.
- Use event-based templating to include observation data from event attributes to add meaningful, actionable context.
- Keep your Sumo Logic HTTP Logs and Metrics Source URL secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Sumo Logic Handler plugin, use the Sensu Catalog in the web UI to configure and install it. Or, use our curated, configurable quick-start templates for event storage and metric storage to integrate Sensu with your existing workflows and send observation data to an HTTP Logs and Metrics Source.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Sumo Logic Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

Read Send data to Sumo Logic with Sensu to learn how to add and configure a handler that uses the Sensu Sumo Logic Handler plugin.

TimescaleDB integration

The Sensu TimescaleDB Handler plugin is a Sensu handler that sends Sensu metrics to the time-series database TimescaleDB so you can store, instrument, and visualize Sensu metrics data.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Transform metrics to TimescaleDB format: extract and transform the metrics you collect from different sources in formats like Graphite, OpenTSDB, Nagios, and Influx and populate them into TimescaleDB.
- Specify custom values for Sensu event metric points via metric tags.

Get the plugin

For a turnkey experience with the Sensu TimescaleDB Handler plugin, use the Sensu Catalog in the web UI to configure and install it.

To build your own workflow or integrate Sensu with existing workflows, you can also add the Sensu TimescaleDB Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Wavefront integration

The Sensu Wavefront Handler plugin is a Sensu handler that sends Sensu metrics to Wavefront via a proxy, which allows you to store, instrument, and visualize Sensu metrics data in an Wavefront database.

***PRO TIP**: Use the Sensu Catalog to enable this integration directly from your browser. Follow the Catalog prompts to configure the Sensu resources you need and start processing your observability data with a few clicks.*

Features

- Transform metrics to Wavefront format: extract and transform the metrics you collect from different sources in formats like Graphite, OpenTSDB, Nagios, and Influx and populate them into Wavefront.
- Specify additional tags to include when processing metrics with the Wavefront plugin's `tags` flag or metric tags.
- Keep your Graphite host and port secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Wavefront Handler plugin, use the Sensu Catalog in the web UI to configure and install it. Or, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in Wavefront.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Wavefront Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Learn Sensu

The Learn Sensu category includes tools to help you understand and start using Sensu, the industry-leading observability pipeline for multi-cloud monitoring, consolidating monitoring tools, and filling observability gaps at scale.

Concepts and terminology

If you're new to Sensu, start with a basic review of Sensu concepts and terminology, which includes definitions and links to relevant reference documentation for more in-depth information.

To visualize how Sensu concepts work together in the observability pipeline, take the tour — follow the `Next` buttons on each page.

Sensu Go workshop

The Sensu Go workshop is a collection of resources designed to help you learn Sensu:

- Interactive lessons designed for self-guided learning.
- Detailed instructions for Linux, macOS, and Windows workstations.
- A local sandbox environment for use with the workshop (via Docker Compose or Vagrant)

Additional workshop materials are available for advanced use cases, including instructor-led workshops with a multi-tenant sandbox environment and alternative sandbox environments based on popular Sensu reference architectures like InfluxDB, TimescaleDB, Elasticsearch, and Prometheus.

Follow the workshop lessons to build your first observability workflow with Sensu.

Live demo

Explore a live demo of the Sensu web UI: view the Entities page to learn what Sensu is monitoring, the Events page for the latest observability events, and the Checks page for active service and metric checks. The live demo also gives you a chance to try commands with sensuctl, the Sensu command

line tool.

Monitor containers and applications

Follow the instructions for Getting Started with Sensu Go on Kubernetes to deploy a Sensu cluster and an example application (NGINX) into Kubernetes with a Sensu agent sidecar. You'll also learn to use sensuctl to configure Nagios-style monitoring checks to monitor the example application with a Sensu sidecar.

Glossary of Sensu concepts and terminology

Agent

A lightweight client that runs on the infrastructure components you want to monitor. Agents self-register with the backend, send keepalive messages, and execute monitoring checks. Each agent belongs to one or more subscriptions that determine which checks the agent runs. An agent can run checks on the entity it's installed on or connect to a remote proxy entity. Read more about the Sensu agent.

Asset

An executable that a check, handler, or mutator can specify as a dependency. Dynamic runtime assets must be a tar archive (optionally gzipped) with scripts or executables within a bin folder. At runtime, the backend or agent installs required assets using the specified URL. Dynamic runtime assets let you manage runtime dependencies without using configuration management tools. Read more about dynamic runtime assets.

Backend

A flexible, scalable observability pipeline. The Sensu backend processes observation data (events) using filters, mutators, and handlers. It maintains configuration files, stores recent observation data, and schedules monitoring checks. You can interact with the backend using the API, command line, and web UI interfaces. Read more about the Sensu backend.

Business service monitoring (BSM)

A feature that provides high-level visibility into the current health of your business services. An example business service is a company website, which might require several individual elements to have OK status for the website to function (e.g. webservers, an inventory database, and a shopping cart). With business service monitoring (BSM), you could create a current status page for the company website

that displays the website's overall status at a glance.

BSM requires two resources that work together to achieve top-down monitoring: service components and rule templates. Service components are the elements that make up your business services. Rule templates define the monitoring rules that produce events for service components based on customized evaluation expressions.

Read more about BSM, rule templates, and service components.

Catalog

The Sensu Catalog is an element of the Sensu web UI where you can find and install monitoring and observability integrations. An integration combines a Sensu plugin with a dynamic runtime asset and the Sensu resource definitions that use the plugin. The Sensu Catalog includes integrations for standard system checks and metrics collection as well as pipelines for sending Sensu data to third-party logging, remediation, and incident management services. Read more about the Sensu Catalog.

Check

A recurring check the agent runs to determine the state of a system component or collect metrics. The backend is responsible for storing check definitions, scheduling checks, and processing observation data (events). Check definitions specify the command to be executed, an interval for execution, one or more subscriptions, and one or more handlers to process the resulting event data. Read more about checks.

Entity

Infrastructure components that you want to monitor. Each entity runs an agent that executes checks and creates events. Events can be tied to the entity where the agent runs or a proxy entity that the agent checks remotely. Read more about entities.

Event

A representation of the state of an infrastructure component at a point in time. The Sensu backend uses events to power the observability pipeline. Observation data in events include the result of a check or metric (or both), the executing agent, and a timestamp. Read more about events.

Event filter

Logical expressions that handlers evaluate before processing observability events. Event filters can instruct handlers to allow or deny matching events based on day, time, namespace, or any attribute in the observation data (event). Read more about event filters.

Handler

A component of the observability pipeline that acts on events. Handlers can send observability data to an executable (or handler plugin), a TCP socket, or a UDP socket. Read more about handlers.

Hook

A command the agent executes in response to a check result *before* creating an observability event. Hooks create context-rich events by gathering relevant information based on check status. Read more about hooks.

Mutator

An executable the backend runs prior to a handler to transform observation data (events). Read more about mutators.

Pipeline

Resources composed of observation event processing workflows made up of filters, mutators, and handlers. Instead of specifying filters and mutators in handler definitions, you can specify all three in a single pipeline workflow. Read more about pipelines.

Plugin

Executables designed to work with Sensu observation data (events) either as a check, mutator, or

handler plugin. You can write your own check executables in Go, Ruby, Python, and more, or use one of more than 200 plugins shared by the Sensu community. Read more about plugins.

Proxy entities

Components of your infrastructure that can't run the agent locally (like a network switch or a website) but still need to be monitored. Agents create events with information about the proxy entity in place of the local entity when running checks with a specified proxy entity ID. Read more about proxy entities.

Role-based access control (RBAC)

Sensu's local user management system. RBAC lets you manage users and permissions with namespaces, users, roles, and role bindings. Read more about RBAC.

Resources

Objects within Sensu that you can use to specify access permissions in Sensu roles and cluster roles. Resources can be specific to a namespace (like checks and handlers) or cluster-wide (like users and cluster roles). Read more about resources.

Sensuctl

The Sensu command line tool that lets you interact with the backend. You can use sensuctl to create checks, view events, create users, manage clusters, and more. Read more about sensuctl.

Silencing

Entries that allow you to suppress execution of event handlers on an ad-hoc basis. Use silencing to schedule maintenance without being overloaded with alerts. Read more about silencing.

Subscriptions

Attributes used to indicate which entities will execute which checks. For Sensu to execute a check, the check definition must include a subscription that matches the subscription of at least one Sensu entity. Subscriptions allow you to configure check requests in a one-to-many model for entire groups or subgroups of entities rather than a traditional one-to-one mapping of configured hosts or observability checks. Read more about subscriptions.

Token

A placeholder in a check definition that the agent replaces with local information before executing the check. Tokens let you fine-tune check attributes (like thresholds) on a per-entity level while reusing the check definition. Read more about tokens.

Live demonstration of Sensu

Try a live demo of the Sensu web UI. Log in with username `guest` and password `i<3sensu`.

Explore the Entities page to learn what Sensu is monitoring, the Events page for the latest observability events, and the Checks page for active service and metric checks.

You can also use the demo to try out sensuctl, the Sensu command line tool. First, install sensuctl on your workstation. Then, configure sensuctl to connect to the demo.

Run `sensuctl configure` and enter the following information:

```
Authentication method: username/password
Sensu Backend API URL: https://caviar.tf.sensu.io:8080
Namespace: default
Preferred output format: tabular
Username: guest
Password: i<3sensu
```

With sensuctl configured, to view the latest observability events, run:

```
sensuctl event list
```

Read the sensuctl documentation to get started using sensuctl.

About the demo

The Caviar project shown in the demo monitors the Sensu docs site using a licensed Sensu cluster of three backends.