Sensu Go

Contents

| Learn about licensing

Sensu is the industry-leading solution for multi-cloud monitoring at scale. The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud environments. Founded in 2017, Sensu offers a comprehensive monitoring solution for enterprises, providing complete visibility across every system, every protocol, every time — from Kubernetes to bare metal. **Get started now and feel the #monitoringlove:** Learn Sensu Go.

Sensu Go is the latest version of Sensu, designed to be more portable, easier and faster to deploy, and (even more) friendly to containerized and ephemeral environments. Learn about support packages and commercial features designed for monitoring at scale.

Automate your monitoring workflows: Limitless pipelines let you validate and correlate events, mutate data formats, send alerts, manage incidents, collect and store metrics, and more.

Sensu Event Pipeline

Event Application error rate >10%	→	Filter during business hours	→	Mutator ---	→	Handler send to #monitoring in Slack
Event StatsD metric	→	Filter in production	→	Mutator add metric tags	→	Handler store in InfluxDB
Event Server process stopped	→	Filter on state change	→	Mutator ---	→	Handler create incident in PagerDuty
Event User login failure	→	Filter after 10 events	→	Mutator ---	→	Handler create JIRA issue

Reduce alert fatigue: Sensu gives you full control over your alerts with flexible <u>filters</u>, <u>context-rich notifications</u>, reporting, <u>event handling</u>, and auto-remediation.

Integrate anywhere: Sensu's open architecture makes it easy to integrate monitoring with tools you already use like Nagios plugins, Chef, Graphite, InfluxDB, and PagerDuty.

<u>Listen to Sensu Inc. CEO Caleb Hailey explain the Sensu monitoring event pipeline.</u>

Monitoring for your infrastructure

> Monitoring is the action of observing and checking the behaviors and outputs of a system and its components over time. - <u>Greg Poirier, Monitorama 2016</u>
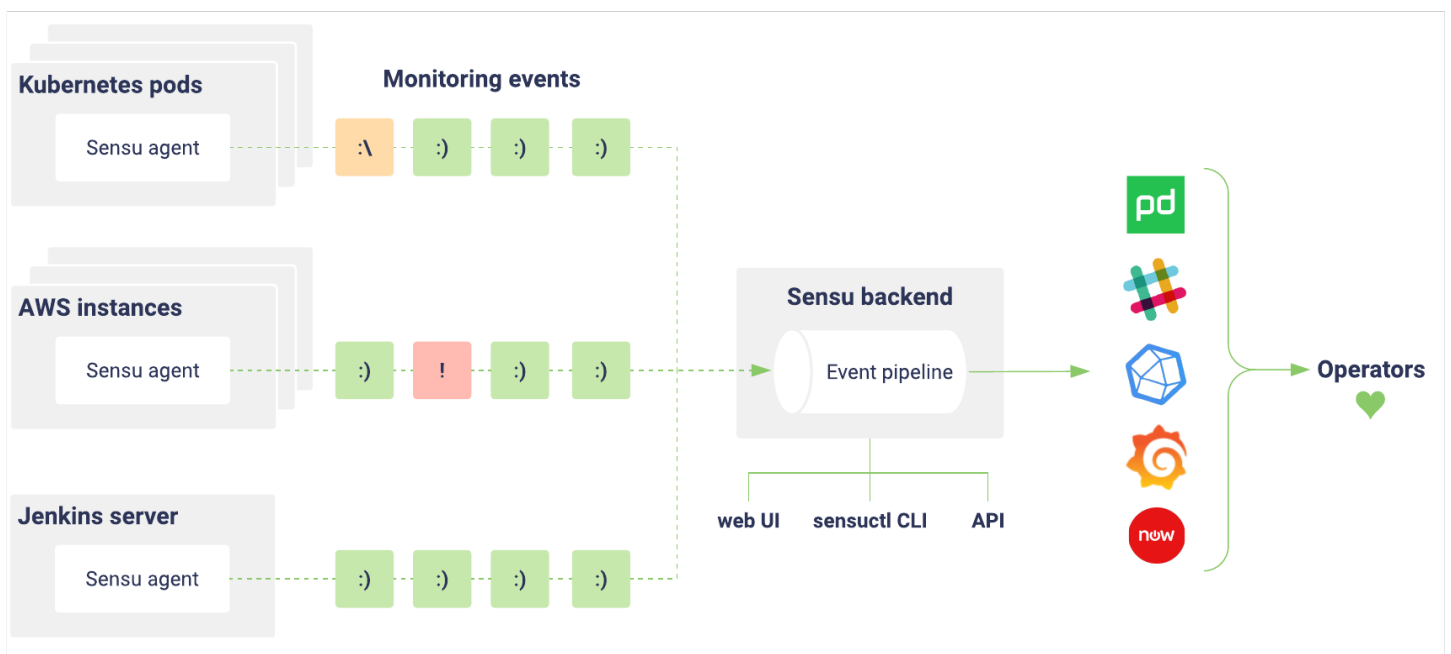
Sensu is an agent-based monitoring tool that you install on your organization's infrastructure. The Sensu agent gives you visibility into everything you care about. The Sensu backend gives you flexible, automated workflows to route metrics and alerts.

Monitor containers, instances, applications, and on-premises infrastructure

Sensu is designed to monitor everything from the server closet to the cloud. Install the Sensu agent on the hosts you want to monitor, integrate with the Sensu API, or take advantage of proxy entities to monitor anything on your network. Sensu agents automatically register and de-register themselves with the Sensu backend, so you can monitor ephemeral infrastructure without getting overloaded with alerts.

Better incident response with filterable, context-rich alerts

Get meaningful alerts when and where you need them. Use event filters to reduce noise and check hooks to add context and speed up incident response. Sensu integrates with the tools and services your organization already uses like PagerDuty, Slack, and more. Check out Bonsai, the Sensu asset index, or write your own Sensu plugins in any language.

Collect and store metrics with 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, and StatsD metrics. Use the Sensu agent to collect metrics alongside check results, then use the event pipeline to route the data to a time series database like InfluxDB.

Intuitive API and dashboard interfaces

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

Open core software backed by Sensu Inc.

Sensu Go's core is open source software, freely available under a permissive MIT License and publicly available on GitHub. Learn about support packages and commercial features designed for monitoring a scale.

Sensu Go 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.

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.

See 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 flags to help you take advantage of etcd auto-discovery features and agent flags 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.

See the upgrade guide to upgrade Sensu to version 5.16.0.

IMPORTANT:

- For Ubuntu/Debian and RHEL/CentOS 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 flag, `--etcd-client-urls`, which should be used with sensu-backend when it is not operating as an etcd member. The flag is also used by the new `sensu-backend init` subcommand.

- Added the 'sensu-backend init' subcommand.

- Added the `--etcd-discovery` and `--etcd-discovery-srv` flags to sensu-backend, which allow users to take advantage of the embedded etcd's auto-discovery features.

- Added `--keepalive-critical-timeout` to define the time after which a critical keepalive event should be created for an agent and `--keepalive-warning-timeout`, 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.

See 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 up to an entity limit of 100. You will see 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. See 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 CentOS 8.

See the <u>upgrade guide</u> to upgrade Sensu to version 5.14.2.

IMPROVEMENTS:

- Upgraded etcd to 3.3.17.
- Added build package for CentOS 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 `--no-embed-etcd` .
- 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.

See the <u>upgrade guide</u> 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.

See 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.

See 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.

See 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 Go asset index, making it easier than ever to fetch and use countless Sensu monitoring plugins and integrations.Additionally, sensuctl now supports loading resource configuration files (e.g. 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!

See the upgrade guide to upgrade Sensu to version 5.13.0.

NEW FEATURES:

- ⌐ Sensuctl now integrates with Bonsai, the Sensu Go asset index.Run a single sensuctl

command to add an asset to your Sensu cluster (e.g. `sensuctl asset add sensu/sensu-pagerduty-handler:1.1.0`).Check to see which assets are outdated (new releases available) with the `outdated` subcommand (e.g. `sensuctl asset outdated`).

- ◦ Sensuctl now supports the `env` subcommand for exporting sensuctl configuration and API tokens to your shell environment (e.g. `eval $(sensuctl env)`).

- ◦ Sensuctl now supports loading multiple resource configuration files (e.g. checks and handlers) from directories!Sensuctl can also load a file using a URL (e.g. `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.

See 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.See the installation guide for the latest download links.

NEW FEATURES:

- ◦ Operators can now authenticate to Sensu via OpenID Direct Connect (OIDC) using sensuctl. See our 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 whitelist of executables that are allowed to run as check and hook commands.See 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.

See 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.

See 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` .See 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.See the asset reference for examples.

- Sensu agents now support the `disable-assets` configuration flag, allowing you to disable asset retrieval for individual agents.See 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.See 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, see the platform 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 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 CentOS/RHEL 5. As a result, CentOS/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.

See 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.

See 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.

See 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, see the web UI docs.

- ⌐ Free Sensu instances can now delete entities in the web UI entities page. See the 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. See 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. See the cluster API 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. See 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. See 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 (e.g. `@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.

See the upgrade guide to upgrade Sensu to version 5.9.0. If you're upgrading a Sensu cluster from 5.7.0 or earlier, see 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. See the 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. See the 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 flags. You can use this event log file as an input source for your favorite data lake solution. See 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). See the <u>supported platforms page</u> for a complete list of Sensu's supported platforms and the <u>installation guide</u> 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.See the <u>events reference</u> 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.

See the <u>upgrade guide</u> 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.See the <u>upgrade guide</u> for more information.

IMPROVEMENTS:

- ⌐ The sensuctl command line tool now supports the `--chunk-size` flag to help you handle large datasets.See the <u>sensuctl reference</u> 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.See the <u>backend reference</u> for more information.

- ⌐ The Sensu API now includes the version API, returning version information for your Sensu

instance.See 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.See the Tessen reference for more information.

- ¬ Sensu licenses now include the entity limit attached to your Sensu licensing package.See 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.See 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 users and namespaces APIs 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.

See 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.See 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 <u>label and field selectors</u> 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.

See the <u>upgrade guide</u> to upgrade Sensu to version 5.6.0.

NEW FEATURES:

- (<u>Commercial feature</u>) Manage your Sensu checks from your browser: Sensu's web user interface now supports creating, editing, and deleting checks.See the <u>docs</u> to get started using the Sensu web UI.

- (<u>Commercial feature</u>) The Sensu web UI now includes an option to delete entities.

- (<u>Commercial feature</u>) 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.See the <u>API docs</u> and <u>sensuctl reference</u> for usage examples.

IMPROVEMENTS:

- (<u>Commercial feature</u>) Sensu's LDAP and Active Directory integrations now support mutual authentication using the `trusted_ca_file`, `client_cert_file`, and `client_key_file` attributes.See the <u>guide to configuring an authentication provider</u> for more information.

- (<u>Commercial feature</u>) Sensu's LDAP and Active Directory integrations now support connecting to an authentication provider using anonymous binding.See the <u>LDAP</u> and <u>Active Directory</u> binding configuration docs to learn more.

- The <u>health API</u> 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"` . See the <u>agent reference</u> 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.

See the <u>upgrade guide</u> 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. See the <u>agent reference</u> to add annotations in the agent configuration file.
- ¬ The <u>agent socket event format</u> 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 <u>blog post</u> on Tessen.

See the <u>upgrade guide</u> to upgrade Sensu to version 5.5.0.

NEW FEATURES:

- ¬ Tessen, the Sensu call-home service, is now enabled by default in Sensu backends.See 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.

See 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.See 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.See the API overview for more information.
- ¬ Sensu now surfaces internal metrics using the metrics API.See the metrics API reference for more information.

IMPROVEMENTS:

- ¬ Sensu now lets you specify a separate TLS certificate and key to secure the dashboard.See the backend reference to configure the `dashboard-cert-file` and `dashboard-key-file` flags,

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. See the [agent reference](#) for more information.

FIXES:

- The backend now processes events without persisting metrics to etcd.

- The events API POST and PUT endpoints now add the current timestamp to new events by default.

- The users API now returns 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 (e.g. `--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.

See 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` .See the check reference for more information.

- Sensu now provides <u>commercial</u> support for using Microsoft Active Directory as an external authentication provider.Read the <u>authentication guide</u> to configure Active Directory, and check out the <u>getting started guide</u> 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.See the <u>agent reference</u> 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` flag 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.

See 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, see our blog post. 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.

See 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. See 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. See the getting started guide.

-

- Sensu now provides the option to limit check output size or to drop check outputs following metric extraction.See the checks reference for more information.

IMPROVEMENTS:

- Sensu now includes support for Debian 8 and 9.See 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` .See 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.See 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.

See 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.See the access control reference and API docs for more information.

IMPROVEMENTS:

- The Sensu agent and sensuctl tool are now available for 32-bit Windows.See 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.See 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.

See 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` .See the upgrade guide for more information.

NEW FEATURES:

- ￢ Sensu agents now include `trusted-ca-file` and `insecure-skip-tls-verify` configuration flags, 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.

See 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, see 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:

- ⌐ Download the sandbox.

- ⌐ Install Sensu Go.

- ⌐ Get started monitoring server resources.

Get started with Sensu

Learn Sensu in 15 minutes

Get started with Sensu, and create your first monitoring event pipeline.

- ⌐ [Get started in your browser with an interactive tutorial](#)

- ⌐ [Download the sandbox](#)

- ⌐ [Check out the live demo](#)

Install Sensu Go (free)

Sensu Go is the flexible monitoring event pipeline designed for container-based and multi-cloud infrastructures.Get started for free on up to 100 entities by installing an official Sensu distribution.

- ⌐ [Install Sensu Go](#)

- ⌐ [Discover Sensu assets](#)

- ⌐ [Learn about free features](#)

Monitor at scale (commercial features)

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

- ⌐ [Learn about commercial features](#)

- ⌐ [Contact the sales team for a free trial](#)

- ⌐ [Activate your Sensu commercial license](#)

Build from source (OSS)

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

- Visit Sensu Go on GitHub
- Learn about OSS features
- Build from source

Get started with commercial features

Sensu Go offers commercial features designed for monitoring at scale. Contact the Sensu sales team for a personalized demo and free trial. All commercial features are available for free in the packaged Sensu Go distribution up to an entity limit of 100. See the announcement on our blog for more information.

Commercial features in Sensu Go

- **Manage resources from your browser**: Create, edit, and delete checks, handlers, mutators, and filters using the Sensu web UI, and access the Sensu web UI homepage.

- **Authentication providers**: Scale Sensu role-based access control (RBAC) with LDAP and Active Directory integrations.

- **Scalable resource filtering** designed for large installations: Use label and field selectors to filter Sensu API responses, sensuctl outputs, and Sensu web UI views using custom labels and a wider range of resource attributes.

- **Event logging**: Log event data to a file you can use as an input source for your favorite data lake solution. The event logging functionality provides better performance and reliability than event handlers.

- **Enterprise-tier assets**: Connect your monitoring event pipelines to industry-standard tools like ServiceNow and Jira with enterprise-tier assets.

- **Enterprise-scale event storage**: Scale your Sensu instance and handle high volumes of events with a PostgreSQL event store.

- **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.

Contact us for a free trial

For a personalized demo and free trial of commercial features at scale in Sensu Go, contact the Sensu sales team. Manage your Sensu account and contact support through account.sensu.io.

- Contact the Sensu sales team

- Log in to your Sensu account

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.

Log in to your Sensu account at account.sensu.io and download your commercial license file.Click **Download license**.

Sensu Go License

View and download your Sensu Go license key.

Account ID

44

Billing Email

Issued

February 19, 2019

Expires

February 19, 2020

Download license

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

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

Use sensuctl to view your license details at any time:

```
sensuctl license info
```

These resources will help you get started with commercial features in Sensu Go:

- ⌐ [Set up and manage authentication providers](#)
- ⌐ [Install plugins with assets](#)
- ⌐ [Manage your Sensu commercial license](#)
- ⌐ [Log in to your Sensu account](#)
- ⌐ [Contact Sensu support](#)

Live demo of Sensu

See a live demo of the Sensu dashboard. Log in with username `guest` and password `i<3sensu`.

Explore the Entities page to see what Sensu is monitoring, the Events page to see the latest monitoring events, and the Checks page to see 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:

```
sensuctl configure
? Sensu Backend URL: https://caviar.tf.sensu.io:8080
? Username: guest
? Password: i<3sensu
? Namespace: default
? Preferred output format: tabular
```

With sensuctl configured, to see the latest monitoring events, run:

```
sensuctl event list
```

See the sensuctl quickstart 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.

Sensu sandbox

Welcome to the Sensu sandbox! The sandbox is the best place to get started with Sensu and try out new features.

Learn Sensu

¬ Download the Sensu sandbox and build your first monitoring workflow

Monitor containers and applications

¬ Deploy a Sensu Go cluster and example app with Kubernetes and monitor the app with Sensu

Collect metrics

¬ Collect Prometheus metrics with Sensu

Upgrade from Sensu Core 1.x to Sensu Go

¬ Use the Sensu translator to translate check configurations from Sensu Core 1.x to Sensu Go

Glossary of Sensu terms

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. 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. Assets let you manage runtime dependencies without using configuration management tools. Read more about assets.

Backend

A flexible, scalable monitoring event pipeline. The Sensu backend processes event data using filters, mutators, and handlers. It maintains configuration files, stores recent event data, and schedules monitoring checks. You can interact with the backend using the API, command line, and dashboard interfaces. Read more about the Sensu backend.

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 event data. 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 monitoring event pipeline. Event data includes 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 monitoring events. Event filters can instruct handlers to allow or deny matching events based on day, time, namespace, or any attribute in the event data. Read more about event filters.

Handler

A component of the monitoring event pipeline that acts on events. Handlers can send monitoring event 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 a monitoring 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 event data. Read more about mutators.

Plugin

Executables designed to work with Sensu event data 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.

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.

Frequently asked questions about Sensu

Thank you for visiting the Sensu FAQ! For a list of Sensu terms and definitions, see the glossary.

- ⌐ What platforms does Sensu support?
- ⌐ Is Sensu available as a hosted solution?
- ⌐ What are the hardware requirements for running a Sensu backend?
- ⌐ Is there an enterprise version of Sensu Go?
- ⌐ What's the difference between the OSS, free, and commercial versions?
- ⌐ How can I contact the Sensu sales team?
- ⌐ What can I monitor with Sensu?
- ⌐ Does Sensu include a time series database for long-term storage?
- ⌐ Can I connect Sensu Go to clients and servers from earlier versions of Sensu Core and Sensu Enterprise?
- ⌐ Can I upgrade my Sensu Core 1.x deployment to Sensu Go?
- ⌐ Which ports does Sensu use?
- ⌐ Can one Sensu backend monitor multiple sites?
- ⌐ Can I use Uchiwa with Sensu Go?

What platforms does Sensu support?

Sensu Go is available for Linux, Windows (agent and CLI only), macOS (CLI only), Solaris, and Docker. See the list of supported platforms and the installation guide for more information.

Is Sensu available as a hosted solution?

No, Sensu is installed on your organization's infrastructure alongside other applications and services. See the list of supported platforms and the installation guide for more information.

What are the hardware requirements for running a Sensu backend?

See the hardware requirements guide for minimum and recommended hardware to run a Sensu backend.

Is there an enterprise version of Sensu Go?

Yes! Sensu Inc. offers support packages for Sensu Go as well as commercial features designed for monitoring at scale. Contact the Sensu sales team for a personalized demo. See Get started with commercial features for more information.

What's the difference between the OSS, free, and commercial versions?

See the Enterprise page for a complete comparison.

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

How can I contact the Sensu sales team?

We'd love to chat about solving your organization's monitoring challenges with Sensu. Get in touch with us using this form.

What can I monitor with Sensu?

Sensu supports a wide range of plugins for monitoring everything from the server closet to the cloud. Install the Sensu agent on the hosts you want to monitor, integrate with the Sensu API, or take advantage of proxy entities to monitor anything on your network.

Sensuctl integrates with Bonsai, the Sensu asset index, where you'll find plugins, libraries, and

runtimes you need to automate your monitoring workflows. If you want to add your own asset, read the [guide for sharing an asset on Bonsai](#).

You can also check out the 200+ plugins shared in the [Sensu plugins community](#), including monitoring checks for [AWS](#), [Jenkins](#), [Puppet](#), [InfluxDB](#), and [SNMP](#), or write your own Sensu plugins in any language using the [Sensu plugin specification](#).

Does Sensu include a time series database for long-term storage?

No, Sensu does not store event data. We recommend integrating Sensu with a time series database, like [InfluxDB](#), to store event data. See the [guide to storing metrics with InfluxDB](#) to get started.

Can I connect Sensu Go to clients and servers from earlier versions of Sensu Core and Sensu Enterprise?

No, Sensu Go agents and backends are not compatible with Sensu Core or Sensu Enterprise services.

Can I upgrade my Sensu Core 1.x deployment to Sensu Go?

Sensu Go is a complete redesign of the original Sensu. It uses separate packages, dependencies, and data models to bring you powerful new features. See the [Sensu Go release announcement](#) for more information. Due to these changes, [some Sensu Core 1.x features](#) are no longer supported in Sensu Go, such as standalone checks. To upgrade your Sensu Core 1.x deployment to Sensu Go, you'll need to translate your Sensu Core 1.x configuration to the format Sensu Go expects and install the new Sensu Go services on your infrastructure. The [Sensu Go upgrade guide](#) includes a detailed feature comparison between Sensu Go and Sensu Core 1.x as well as tools to help you get started.

Which ports does Sensu use?

The [Sensu backend](#) uses:

- 2379 (HTTP/HTTPS) Sensu storage client: Required for Sensu backends using an external etcd instance

- 2380 (HTTP/HTTPS) Sensu storage peer: Required for other Sensu backends in a [cluster](#)

- ⌐ 3000 (HTTP/HTTPS) <u>Sensu dashboard</u>: Required for all Sensu backends using a Sensu dashboard

- ⌐ 8080 (HTTP/HTTPS) <u>Sensu API</u>: Required for all users accessing the Sensu API

- ⌐ 8081 (WS/WSS) Agent API: Required for all Sensu agents connecting to a Sensu backend

The <u>Sensu agent</u> uses:

- ⌐ 3030 (TCP/UDP) Sensu <u>agent socket</u>: Required for Sensu agents using the agent socket

- ⌐ 3031 (HTTP) Sensu <u>agent API</u>: Required for all users accessing the agent API

- ⌐ 8125 (UDP, TCP on Windows) <u>StatsD listener</u>: Required for all Sensu agents using the StatsD listener

The agent TCP and UDP sockets are deprecated in favor of the <u>agent API</u>.

For more information, see the <u>Secure Sensu guide</u>.

Can one Sensu backend monitor multiple sites?

Yes, as long as you meet the <u>port requirements</u>, a single Sensu backend can monitor Sensu agents at multiple sites.

Can I use Uchiwa with Sensu Go?

Due to Sensu Go's implementation, it is not possible to use Uchiwa with Sensu Go.Sensu Go does have a <u>built-in dashboard</u> that you can use to visually interact with your Sensu Go deployment.

Sensu Go media

This media guide includes a collection of links to talks, blog posts, tutorials, and podcasts about Sensu Go.

NOTE: *Before October 2018, Sensu Go was known as Sensu 2.0.*

Talks

- Assets in Sensu Go: Greg Poirier, Sensu Summit 2018

- Data Collection and Prometheus Scraping With Sensu Go: Sean Porter, InfluxDays 2018

- Migrating to Sensu Go: Caleb Hailey and Jef Spaleta, Sensu Summit 2019

- Sensu Go Deep Dive: Greg Poirer, Sensu Summit 2017

Blog posts

- Check output metric extraction with InfluxDB & Grafana: Nikki Attea

- Filters: valves for the Sensu monitoring event pipeline: Eric Chlebek

- How to monitor 1,000 network devices using Sensu Go and Ansible: Christian Michel

- Migrating to 2.0: Jef Spaleta

- Self-service monitoring checks in Sensu Go: Sean Porter

- Sensu Go is here!: Anna Plotkin

- Sensu Habitat Core Plans are Here: Greg Schofield

- Understanding RBAC in Sensu Go: Simon Plourde

Tutorials

- Learn Sensu in 15 minutes interactive tutorial

- Sensu sandbox tutorials

Podcasts

- ⌐ [Sensu Community Chat November 2018](#)

Install Sensu

This installation guide describes how to install the Sensu backend, Sensu agent, and sensuctl command line tool.

Sensu Go is available for Linux, Windows (agent and CLI only), macOS (CLI only), and Docker. If you're trying Sensu for the first time, we recommend setting up a local environment using the Sensu sandbox. If you're deploying Sensu to your infrastructure, we recommend one of our supported packages, Docker images, or configuration management integrations. Sensu downloads are provided under the Sensu commercial license. See supported platforms for more information.

Architecture overview
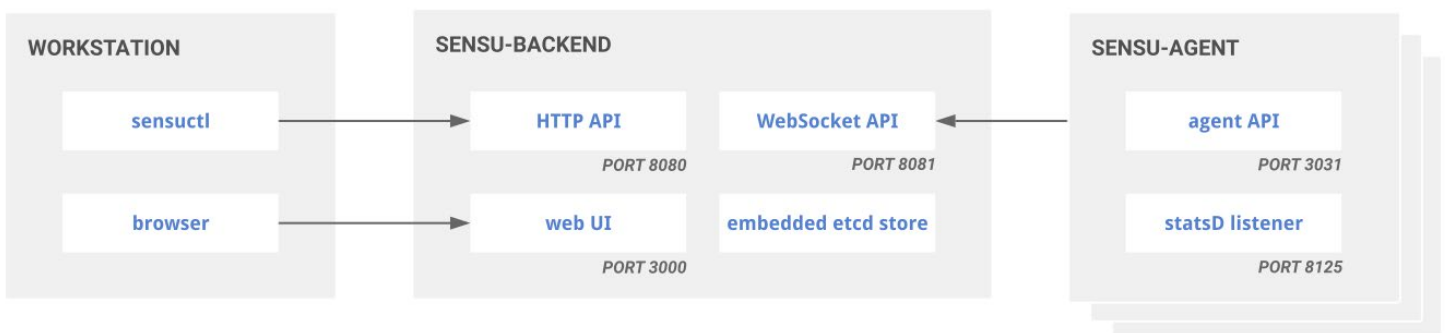
The **Sensu backend** gives you flexible, automated workflows to route metrics and alerts. It is powered by an an embedded transport and etcd datastore and Sensu backends require persistent storage for their embedded database, disk space for local asset caching, and three exposed ports:

- ¬ `3000` - Sensu web UI
- ¬ `8080` - Sensu API used by sensuctl, some plugins, and any of your custom tooling

¬ `8081` - WebSocket API used by Sensu agents

Sensu backends running in a clustered configuration require additional ports. See Deploy Sensu and hardware requirements for deployment recommendations.

Sensu agents are lightweight clients that run on the infrastructure components you want to monitor. Agents register automatically with Sensu as entities and are responsible for creating check and metric events to send to the backend event pipeline. Agents can expose ports `3031` for the agent API and `8125` for the StatsD listener. Agents using Sensu assets require some disk space for a local cache.

Install the Sensu backend

The Sensu backend is available for Ubuntu/Debian, RHEL/CentOS, and Docker. See 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
```

UBUNTU/DEBIAN

```
# 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
```

RHEL/CENTOS

```
# 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.

DOCKER

```
docker run -v /var/lib/sensu:/var/lib/sensu \
-d --name sensu-backend \
-p 3000:3000 -p 8080:8080 -p 8081:8081 sensu/sensu:latest \
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

DOCKER COMPOSE

```
---
version: "3"
services:
  sensu-backend:
    image: sensu/sensu:latest
    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"

volumes:
  sensu-backend-data:
    driver: local
```

UBUNTU/DEBIAN

```
# 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 service sensu-backend start

# Verify that the backend is running
service sensu-backend status
```

RHEL/CENTOS

```
# 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 service sensu-backend start

# Verify that the backend is running
service sensu-backend status
```

For a complete list of configuration options, see the backend reference.

3. Initialize

NOTE: If you are using Docker, skip this step and continue with 4. Open the web UI. The `sensu-backend init` *command is not implemented for Docker.*

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 `YOUR_USERNAME` and `YOUR_PASSWORD` with the username and password you want to use:

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=YOUR_USERNAME
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=YOUR_PASSWORD
```

```
sensu-backend init
```

For details about `sensu-backend init`, see the backend reference.

4. Open the web UI

The web UI provides a unified view of your monitoring 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 (the user ID and password you provided with `sensu-backend init`, or `admin` and `P@ssw0rd!` if you're using Docker). 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 see 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:

UBUNTU/DEBIAN

```
# 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
```

RHEL/CENTOS

```
# 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
```

WINDOWS

```
# Download sensuctl for Windows amd64
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.16.1/sensu-
go_5.16.1_windows_amd64.zip  -OutFile C:\Users\Administrator\sensu-
go_5.16.1_windows_amd64.zip


# Or for 386
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.16.1/sensu-
go_5.16.1_windows_386.zip  -OutFile C:\Users\Administrator\sensu-
go_5.16.1_windows_386.zip
```

MACOS

```
# Download the latest release
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.16.1/sensu-
go_5.16.1_darwin_amd64.tar.gz


# Extract the archive
tar -xvf sensu-go_5.16.1_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 see your user profile.

We recommend that you change the default admin password immediately: `sensuctl user change-password --interactive`.

For more information about sensuctl, see the quickstart and reference docs.

Install Sensu agents

The Sensu agent is available for Ubuntu/Debian, RHEL/CentOS, Windows, and Docker. See 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
```

UBUNTU/DEBIAN

```
# 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
```

RHEL/CENTOS

```
# 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
```

WINDOWS

```
# Download the Sensu agent for Windows amd64
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.16.1/sensu-
go-agent_5.16.1.8521_en-US.x64.msi  -OutFile "$env:userprofile\sensu-go-
agent_5.16.1.8521_en-US.x64.msi"

# Or for Windows 386
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.16.1/sensu-
go-agent_5.16.1.8521_en-US.x86.msi  -OutFile "$env:userprofile\sensu-go-
agent_5.16.1.8521_en-US.x86.msi"

# Install the Sensu agent
msiexec.exe /i $env:userprofile\sensu-go-agent_5.16.1.8521_en-US.x64.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 COMPOSE

```
# 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
```

UBUNTU/DEBIAN

```
# 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
service sensu-agent start
```

RHEL/CENTOS

```
# 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
service sensu-agent start
```

WINDOWS

```
# 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
```

For a complete list of configuration options, see the agent reference.

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`.

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, <u>log in to your Sensu account</u> and download your license file, then add your license using sensuctl.

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

You can use sensuctl to view your license details at any time.

```
sensuctl license info
```

Next steps

Now that you've installed Sensu, here are some resources to help continue your journey:

- <u>Send Slack alerts with handlers</u>
- <u>Monitor server resources with checks</u>
- <u>Aggregate metrics with the Sensu StatsD listener</u>
- <u>Create a read-only user with RBAC</u>

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 assets

Assets are shareable, reusable packages that make it easier to deploy Sensu plugins. To start using and deploying assets, read Install plugins with assets to become familiar with workflows that involve assets.

Use Bonsai, the Sensu asset index

Bonsai, the Sensu asset index, is a centralized place for downloading and sharing plugin assets. Make Bonsai your first stop when you need to find an asset. Bonsai includes plugins, libraries, and runtimes you need to 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`.

```
sensu-install --help
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'
```

We recommend using a configuration management tool or using Sensu 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 this discourse.sensu.io guide 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. Depending on the plugin, you may need to install developer tool packages.

Ubuntu/Debian:

```
sudo apt-get update
```

```
sudo apt-get install build-essential
```

RHEL/CentOS:

```
sudo yum update
```

```
sudo yum groupinstall "Development Tools"
```

Upgrade Sensu

Upgrade to the latest version of Sensu Go from 5.0.0 or later

To upgrade to the latest version of Sensu Go from version 5.0.0 or later, [install the latest packages](#).

Then, restart the services.

NOTE: *For systems that use* `systemd` , *run* `sudo systemctl daemon-reload` *before restarting the services.*

```
# Restart the Sensu agent
sudo service sensu-agent restart

# Restart the Sensu backend
sudo service sensu-backend restart
```

Use the `version` command to determine the installed version using the `sensu-agent` , `sensu-backend` , and `sensuctl` tools. For example, `sensu-backend version` .

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 up to an entity limit of 100.

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 see 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 see the following message when you reach the 100-entity limit:

⚠ We noticed you've reached the free usage limit of 100 entities. You are not able to create additional entities beyond the limit. CONTACT
Please reach out to our sales team to learn how to upgrade your installation. SALES ↗

If your Sensu instance includes more than 100 entities, contact Sales to learn how to upgrade your installation and increase your limit.See 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.See 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`, add the following configuration to `/etc/sensu/backend.yml`.

```
# /etc/sensu/backend.yml configuration to store backend data at /var/lib/sensu
state-dir: "/var/lib/sensu"
```

Then restart the backend.

Migrate to Sensu Go from Sensu Core 1.x

This guide includes general information for migrating your Sensu instance from Sensu Core 1.x to Sensu Go 5.0. For instructions and tools to help you translate your Sensu configuration from Sensu Core 1.x to Sensu Go, see the [Sensu translator project](#) and our [blog post about check configuration upgrades with the Sensu Go sandbox](#).

Sensu Go includes important changes to all parts of Sensu: architecture, installation, resource definitions, the event data model, check dependencies, filter evaluation, and more. Sensu Go also includes many powerful features to make monitoring easier to build, scale, and offer as a self-service tool to your internal customers.

- ¬ [Packaging](#)
- ¬ [Architecture](#)
- ¬ [Entities](#)
- ¬ [Checks](#)
- ¬ [Events](#)
- ¬ [Handlers](#)
- ¬ [Filters](#)
- ¬ [Assets](#)
- ¬ [Role-based access control (RBAC)](#)
- ¬ [Silencing](#)
- ¬ [Token substitution](#)
- ¬

Packaging

Sensu is now 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 1.x: the server is now the backend and the client is now the agent.To learn more about new terminology in Sensu Go, see the glossary.

Architecture

The external RabbitMQ transport and Redis datastore in Sensu Core 1.x are replaced with an embedded transport and etcd datastore in Sensu Go.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.See the backend, agent, and sensuctl reference docs for more information.

Entities

"Clients" are represented within Sensu Go as abstract "entities" that can describe a wider range of system components (for example, network gear, a web server, or a cloud resource).Entities include **agent entities**, which are entities running a Sensu agent, and the familiar **proxy entities**.See the entity reference and the guide to monitoring external resources for more information.

Checks

Standalone checks are not supported in Sensu Go, although you can achieve similar functionality with role-based access control (RBAC), assets, and entity subscriptions.There are also a few changes to check definitions in Sensu Go.The `stdin` check attribute is not supported in Sensu Go, and Sensu Go does not try to run a "default" handler when executing a check without a specified handler.In addition, check subdues are not available in Sensu Go.

Check hooks are a resource type in Sensu Go: you can create, manage, and reuse hooks independently of check definitions.You can also execute multiple hooks for any given response code.

Events

In Sensu Go, all check results are considered events and are processed by event handlers. You can use the built-in incidents filter to recreate the Sensu Core 1.x behavior in which only check results with a non-zero status are considered events.

Handlers

Transport handlers are not supported by Sensu Go, but you can create similar functionality with a pipe handler that connects to a message bus and injects event data into a queue.

Filters

Ruby eval logic from Sensu Core 1.x is replaced with JavaScript expressions in Sensu Go, opening up powerful ways to filter events based on occurrences and other event attributes. As a result, the built-in occurrences event filter in Sensu Core 1.x is not included in Sensu Go, but you can replicate its functionality with the repeated events filter definition.

Sensu Go includes three new built-in event filters: only-incidents, only-metrics, and allow-silencing. Sensu Go does not include a built-in check dependencies filter or a filter-when feature.

Assets

The `sensu-install` tool in Sensu Core 1.x is replaced by assets in Sensu Go. 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. See the installing plugins guide 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, see the RBAC reference and Create a read-only user.

Silencing

Silencing is disabled by default in Sensu Go. You must explicitly enable silencing with the built-in `not_silenced` event filter.

Token substitution

The syntax for token substitution changed to double curly braces in Sensu Go (from triple colons in Sensu Core 1.x).

Aggregates

Check aggregates are supported through the Sensu Go Aggregate Check Plugin (a commercial resource).

API

In addition to the changes to resource definitions, Sensu Go includes a new, versioned API. See the API overview for more information.

Custom attributes

Custom check attributes are not supported in Sensu Go. Instead, Sensu Go allows you to add custom labels and annotations to entities, checks, assets, hooks, filters, mutators, handlers, and silences. See the metadata attributes section in the reference documentation for more information about using labels and annotations (for example, metadata attributes for entities).

Authentication

Sensu requires username and password authentication to access the [Sensu dashboard](#), [API](#), and command line tool ([sensuctl](#)). You can use Sensu's built-in basic authentication provider or configure external authentication providers to authenticate via Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect.

Use built-in basic authentication

Sensu's built-in basic authentication provider allows you to create and manage user credentials (usernames and passwords) with the [users API](#), either directly or using [sensuctl](#). The basic authentication provider does not depend on external services and is not configurable.

After creating users via the basic authentication provider, you can manage their permissions via role-based access control (RBAC). See Create read-only users for an example.

Sensu records basic authentication credentials in etcd.

Use an authentication provider

COMMERCIAL FEATURE: Access authentication providers in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

In addition to built-in authentication and RBAC, Sensu includes commercial support for authentication using external authentication providers, including Microsoft Active Directory (AD) and standards-compliant Lightweight Directory Access Protocol (LDAP) tools like OpenLDAP.

Manage authentication providers

View and delete authentication providers with sensuctl and the authentication providers API. To set up an authentication provider for Sensu, see Configure authentication providers.

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
```

Configure authentication providers

1. Write an authentication provider configuration definition

For standards-compliant LDAP tools like OpenLDAP, see the LDAP configuration examples and specification. For Microsoft AD, see the AD configuration examples and specification.

2. Apply the configuration with sensuctl

Log in to sensuctl as the default admin user and apply the configuration to Sensu:

```
sensuctl create --file filename.json
```

Use sensuctl to verify that your provider configuration was applied successfully:

```
sensuctl auth list

 Type    Name
_____ _____

 ldap   openldap
```

3. Integrate with Sensu RBAC

Now that you've configured an authentication provider, you'll need to configure Sensu RBAC to give those users permissions within Sensu. Sensu RBAC allows you to manage and access users and resources based on namespaces, groups, roles, and bindings. See the RBAC reference for more information about configuring permissions in Sensu and implementation examples.

- **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 namespaces and 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 cluster-wide.

To enable permissions for external users and groups within Sensu, create a set of roles, cluster roles, role bindings, and cluster role bindings that map to the usernames and group names found in your authentication providers.

Make sure to include the group prefix and username prefix when creating Sensu role bindings and cluster role bindings. Without an assigned role or cluster role, users can sign in to the Sensu dashboard but can't access any Sensu resources.

4. Log in to Sensu

After you configure the correct roles and bindings, log in to sensuctl and the Sensu dashboard using your single-sign-on username and password (no prefix required).

LDAP authentication

Sensu offers commercial support for a standards-compliant LDAP tool for authentication to the Sensu dashboard, API, and sensuctl. The Sensu LDAP authentication provider is tested with OpenLDAP. If you're using AD, head to the AD section.

LDAP configuration examples

Example LDAP configuration: Minimum required attributes

YML

```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

```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:
  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

```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"
        }
      }
    ],
```

```
      "groups_prefix": "ldap",
      "username_prefix": "ldap"
   },
   "metadata": {
     "name": "openldap"
   }
 }
```

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
example	`"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
example	`"api_version": "authentication/v2"`

metadata

description	Top-level map that contains the LDAP definition `name` . See the metadata attributes reference for details.
required	true
type	Map of key-value pairs

example	```json
	"metadata": {
	  "name": "openldap"
	}
	``

spec

description	Top-level map that includes the LDAP spec attributes.
required	true
type	Map of key-value pairs

example	```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"
        }
      }
    ],
    "groups_prefix": "ldap",
    "username_prefix": "ldap"
}
```

LDAP spec attributes

servers	
description	An array of <u>LDAP servers</u>for your directory. During the authentication process, Sensu attempts to authenticate using each LDAP server in sequence.
required	true
type	Array
example	`"servers": [ { "host": "127.0.0.1", "port": 636, "insecure": false, "security": "tls", "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem",`

```
"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"
      }
    }
  ]
```

groups_prefix

description	The prefix added to all LDAP groups. Sensu prepends prefixes 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	`"groups_prefix": "ldap"`

username_prefix

description	The prefix added to all LDAP usernames. Sensu prepends prefixes 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	```"username_prefix": "ldap"```

LDAP server attributes

host	
description	LDAP server IP address or FQDN.
required	true
type	String
example	```"host": "127.0.0.1"```

port	
description	LDAP server port.
required	true
type	Integer
default	`389` for insecure connections; `636` for TLS connections
example	

```
  "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	```"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	```"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	`"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	`"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	`"client_key_file": "/path/to/ssl/key.pem"`

binding

description	The LDAP account that performs user and group lookups. If your sever supports anonymous binding, you can omit the `user_dn` or `password` attributes to query the directory without credentials.

required	false
type	Map

example	

```
"binding": {
  "user_dn": "cn=binder,dc=acme,dc=org",
  "password": "YOUR_PASSWORD"
}
```

group_search

description	Search configuration for groups. See the group search attributes for more information.
required	true
type	Map

example	

```
"group_search": {
  "base_dn": "dc=acme,dc=org",
  "attribute": "member",
  "name_attribute": "cn",
  "object_class": "groupOfNames"
}
```

user_search

description	Search configuration for users. See the user search attributes for more information.
required	true
type	Map

example	

```
"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 sever supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String
example	`"user_dn": "cn=binder,dc=acme,dc=org"`

password	
description	Password for the `user_dn` account. If your sever supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String
example	`"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	`"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	`"attribute": "member"`

name_attribute

description	Represents the attribute to use as the entry name.
required	false
type	String
default	`"cn"`

example	
	`"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	`"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	`"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	`"attribute": "uid"`

name_attribute

description	Represents the attribute to use as the entry name
required	false
type	String
default	`"cn"`
example	`"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	`"object_class": "person"`

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	`"name": "openldap"`

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, make sure the LDAP provider was successfully installed using sensuctl.

Authentication errors

This section lists common error messages and possible solutions.

Error message: `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.

Error message: `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.

Error message: `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` specifies a valid **DN** and that its password is correct.

Error message: `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

Error message: `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

Error message: `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.

Error message: `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` attribute 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 issues

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`.

Issue: Permissions are not granted via the LDAP group(s)

During authentication, the LDAP provider will print in the logs all groups found in LDAP (for example, `found 1 group(s): [dev]`. 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 [...]
```

Active Directory (AD) authentication

Sensu offers commercial support for using Microsoft Active Directory (AD) for authentication to the Sensu dashboard, API, and sensuctl. The AD authentication provider is based on the LDAP authentication provider.

Active Directory (AD) configuration examples

Example AD configuration: Minimum required attributes

YML

```
type: ad
api_version: authentication/v2
```

```yaml
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

```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

```yaml
type: ad
api_version: authentication/v2
metadata:
```

```yaml
      name: activedirectory
  spec:
    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

```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"
      }
    }
  ],
  "groups_prefix": "ad",
  "username_prefix": "ad"
},
"metadata": {
  "name": "activedirectory"
}
}
```

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	`"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	`"api_version": "authentication/v2"`

metadata

description	Top-level map that contains the AD definition `name`. See the metadata attributes reference for details.
required	true
type	Map of key-value pairs

example	`"metadata": {`
	`  "name": "activedirectory"`
	`}`

spec

description	Top-level map that includes the AD spec attributes.
required	true
type	Map of key-value pairs
example	

```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"
      }
    }
  ],
  "groups_prefix": "ad",
  "username_prefix": "ad"
}
```

AD spec attributes

servers	
description	An array of <u>AD servers</u> for your directory. During the authentication process, Sensu attempts to authenticate using each AD server in sequence.
required	true
type	Array
example	

```
"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"
```

```
      }
    }
  ]
```

groups_prefix

description	The prefix added to all AD groups. Sensu prepends prefixes 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 <u>role bindings</u> and <u>cluster role bindings</u>.
required	false
type	String
example	`"groups_prefix": "ad"`

username_prefix

description	The prefix added to all AD usernames. Sensu prepends prefixes 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 <u>role bindings</u> and <u>cluster role bindings</u>. Users *do not* need to provide this prefix when logging in to Sensu.
required	false
type	String
example	`"username_prefix": "ad"`

AD server attributes

host

description	AD server IP address or FQDN.
required	true
type	String
example	`"host": "127.0.0.1"`

port

description	AD server port.
required	true
type	Integer
default	`389` for insecure connections; `636` for TLS connections
example	`"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	`"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"`
example	`"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	`"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	
	`"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	`"client_key_file": "/path/to/ssl/key.pem"`

binding

description	The AD account that performs user and group lookups. If your sever 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.
required	false
type	Map

example	`"binding": {`
	`  "user_dn": "cn=binder,cn=users,dc=acme,dc=org",`
	`  "password": "YOUR_PASSWORD"`
	`}`

group_search

description	Search configuration for groups. See the group search attributes for more information.

required	true
type	Map
| example | ```
"group_search": {
  "base_dn": "dc=acme,dc=org",
  "attribute": "member",
  "name_attribute": "cn",
  "object_class": "group"
}
``` |

user_search

description	Search configuration for users. See the user search attributes for more information.
required	true
type	Map
| example | ```
"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. To ensure group membership updates are reflected*

without re-authentication, specify a binding account or enable anonymous binding.

required	false
type	String
example	`"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
example	`"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 sever supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String

example	
	`"user_dn": "cn=binder,cn=users,dc=acme,dc=org"`

password

description	Password for the `user_dn` account. If your sever supports anonymous binding, you can omit this attribute to query the directory without credentials.
required	false
type	String
example	`"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	`"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	`"attribute": "member"`

name_attribute

description	Represents the attribute to use as the entry name.
required	false
type	String
default	`"cn"`
example	`"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"`
example	`"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	`"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	`"attribute": "sAMAccountName"`

name_attribute

description	Represents the attribute to use as the entry name.
required	false
type	String
default	`"displayName"`

example	
	`"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	`"object_class": "person"`

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	`"name": "activedirectory"`

AD troubleshooting

The troubleshooting steps in the LDAP troubleshooting section also apply for AD troubleshooting.

OIDC authentication

Sensu offers commercial support for the OIDC provider for using the OpenID Connect 1.0 protocol (OIDC) on top of the OAuth 2.0 protocol for RBAC authentication.

The Sensu OIDC provider is tested with Okta and PingFederate.

OIDC configuration examples

YML

```yaml
---
type: oidc
api_version: authentication/v2
metadata:
  name: oidc_name
spec:
  additional_scopes:
  - groups
  - email
  client_id: a8e43af034e7f2608780
  client_secret: b63968394be6ed2edb61c93847ee792f31bf6216
  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

```json
{
  "type": "oidc",
  "api_version": "authentication/v2",
  "metadata": {
    "name": "oidc_name"
  },
  "spec": {
```

```
      "additional_scopes": [
          "groups",
          "email"
      ],
      "client_id": "a8e43af034e7f2608780",
      "client_secret": "b63968394be6ed2edb61c93847ee792f31bf6216",
      "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
example	`"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
example	`"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	```
	"metadata": {
	  "name": "oidc_name"
	  }
	}
	``

spec	
description	Top-level map that includes the OIDC spec attributes
required	true
type	Map of key-value pairs

example	```
	"spec": {
	  "additional_scopes": [
	    "groups",
	    "email"
	    ],
	``

```
    "client_id": "a8e43af034e7f2608780",
    "client_secret":
  "b63968394be6ed2edb61c93847ee792f31bf6216",
    "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 `metadata.name` cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`).
required	true
type	String
example	`"name": "oidc_name"`

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
example	`"additional_scopes": ["groups", "email", "username"]`

client_id

description	The OIDC provider application `Client ID`. **NOTE**: Requires _registering an application in the OIDC provider_.
required	true
type	String
example	`"client_id": "1c9ae3e6f3cc79c9f1786fcb22692d1f"`

client_secret

description	The OIDC provider application `Client Secret`. **NOTE**: Requires _registering an application in the OIDC provider_.
required	true
type	String
example	`"client_secret": "a0f2a3c1dcd5b1cac71bf0c03f2ff1bd"`

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	
	``` "redirect_uri": "http://sensu- backend.example.com:8080/api/enterprise/authentication/v2/o idc/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	
	``` "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	
	``` "groups_claim": "groups" ```

groups_prefix

description	The prefix to use to form the final RBAC groups if required.

required	false
type	String
example	`"groups_prefix": "okta"`

username_claim

description	The claim to use to form the final RBAC user name.
required	false
type	String
example	`"username_claim": "person"`

username_prefix

description	The prefix to use to form the final RBAC user name.
required	false
type	String
example	`"username_prefix": "okta"`

Register an OIDC application

To use OIDC for authentication, register Sensu Go as an OIDC application.Use the instructions listed in this section to register an OIDC application for Sensu Go based on your OIDC provider.

- ¬ Okta

Okta

Requirements

- ¬ Access to the Okta Administrator Dashboard

- ¬ Sensu Go 5.12.0 or later (plus a valid commercial license for Sensu Go versions 5.12.0 through 5.14.2)

Create an Okta application

1. From the Okta Administrator Dashboard, select `Applications > Add Application > Create New App` to start the wizard.
2. Select the `Web` platform and `OpenID Connect` sign-in method.
3. In *General Settings*, enter an app name and upload a logo (if desired).
4. In *Configure OpenID Connect*, add the following redirect URI (replace `DASHBOARD_URL` with the URL for your dashboard:
   `{DASHBOARD_URL}/api/enterprise/authentication/v2/oidc/callback` .
5. Click **Save**.
6. Open the *Sign On* page. In the *OpenID Connect ID Token* section, click **Edit**.
7. Enter the following information for the *Groups* claim attribute:

   - ¬ First field: `groups`

   - ¬ Dropdown menu: `Regex`

   - ¬ Second field: `.*`

8. Click **Save**.
9. Assign people and groups in the *Assignments* page.

OIDC provider configuration

1. Add the `additional_scopes` configuration attribute in the OIDC scope and set the value to `[ "groups" ]` :

   - ¬ `"additional_scopes": [ "groups" ]`

2. Add the `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:

```
⌐  "groups_claim": "okta:groups"
```

3. Add the `redirect_uri` configuration attribute in the OIDC scope and set the value to the
    Redirect URI configured at step 4 of Create an Okta application:

```
⌐  "redirect_uri": "
   {BACKEND_URL}/api/enterprise/authentication/v2/oidc/callback"
```

Sensuctl login with OIDC

1. Run `sensuctl login oidc` .

2. If you are using a desktop, a browser will open to `OIDC provider` and 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 following URL:

    ⌐ https://sensu-
       backend.example.com:8080/api/enterprise/authentication/v2/oidc/authorize

Hardware requirements

- Sensu backend requirements

- Sensu agent requirements

- Networking recommendations

- Cloud recommendations

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

See the Backend recommended configuration for production recommendations.

Backend recommended configuration

This 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 (NVMe or 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 roughly one CPU core for every 50 checks per second (including agent keepalives). Most installations are fine with four CPU cores, but 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 this 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 under 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, or ARM CPU (ARMv5 minimum)

- ￢ 128 MB RAM

- ￢ 10 mbps network link

See the Agent recommended configuration for production recommendations.

Agent recommended configuration

This 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

Agent connections

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 you see WebSocket timeouts in the backend logs, you may need to use a more reliable network link between the backend and agents.

Cloud recommendations

AWS

The recommended EC2 instance type and size for Sensu backends running embedded etcd is **M5d.xlarge**. The M5d instance provides four vCPU, 16 GB of RAM, up to 10 gbps network connectivity and a 150-NVMe SSD directly attached to the instance host (optimal for sustained disk IOPS).

Binary-only distributions

In addition to packages, Sensu binary-only distributions are available for Linux, Windows (agent and CLI only), macOS (CLI only), FreeBSD, and Solaris.

Linux

Sensu binary-only distributions for Linux are available for these architectures and formats:

arch	format
`amd64`	`.tar.gz` | `.zip`
`arm64`	`.tar.gz` | `.zip`
`armv5` (agent and CLI)	`.tar.gz` | `.zip`
`armv6` (agent and CLI)	`.tar.gz` | `.zip`
`armv7` (agent and CLI)	`.tar.gz` | `.zip`
`386`	`.tar.gz` | `.zip`

NOTE: *32-bit systems cannot run the Sensu backend reliably, so* `armv5` *,* `armv6` *, and* `armv7` *packages include the agent and CLI only.*

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/5.16.1/sensu-go_5.16.1_linux_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
sha256sum sensu-go_5.16.1_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/5.16.1/sensu-
go_5.16.1_checksums.txt && cat sensu-go_5.16.1_checksums.txt
```

Windows

Sensu binary-only distributions for Windows are available for these architectures and formats:

arch	format
amd64	`.tar.gz` | `.zip`
386	`.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/5.16.1/sensu-
go_5.16.1_windows_amd64.zip  -OutFile "$env:userprofile\sensu-
go_5.16.1_windows_amd64.zip"
```

Generate a SHA-256 checksum for the downloaded artifact:

```
Get-FileHash "$env:userprofile\sensu-go_5.16.1_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/5.16.1/sensu-
go_5.16.1_checksums.txt -OutFile "$env:userprofile\sensu-go_5.16.1_checksums.txt"

Get-Content "$env:userprofile\sensu-go_5.16.1_checksums.txt" | Select-String -Pattern
windows_amd64
```

macOS

Sensu binary-only distributions for macOS are available for these architectures and formats:

arch	format
amd64	.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/5.16.1/sensu-
go_5.16.1_darwin_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
shasum -a 256 sensu-go_5.16.1_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/5.16.1/sensu-
go_5.16.1_checksums.txt && cat sensu-go_5.16.1_checksums.txt
```

Extract the archive:

```
tar -xvf sensu-go_5.16.1_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 these architectures and formats:

arch	format
amd64	.tar.gz | .zip
386	.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/5.16.1/sensu-
go_5.16.1_freebsd_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
sha256sum sensu-go_5.16.1_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/5.16.1/sensu-
go_5.16.1_checksums.txt && cat sensu-go_5.16.1_checksums.txt
```

Solaris

Sensu binary-only distributions for Solaris are available for these architectures and formats:

arch	format
`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/5.16.1/sensu-
go_5.16.1_solaris_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact.

```
sha256sum sensu-go_5.16.1_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/5.16.1/sensu-
go_5.16.1_checksums.txt && cat sensu-go_5.16.1_checksums.txt
```

Next steps

Now that you've installed Sensu:

- ¬ Configure sensuctl
- ¬ Start the Sensu backend
- ¬ Start the Sensu agent
- ¬ Use Sensu to monitor server resources

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 GitHub repo.

The Sensu Go Ansible Collection documentation site includes installation instructions, example playbooks, and module references.

Chef

The Chef cookbook for Sensu is available in the Sensu-Go-Chef GitHub repo.

Contact us for more information about Sensu + Chef.

Puppet

The Puppet Sensu module is available in the Sensu-Puppet GitHub repo.

Sensu partnered with Tailored Automation to enhance the Puppet module with new features and bug fixes.

Supported platforms

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

Supported packages are available through sensu/stable on packagecloud and the downloads page.

Sensu backend

Platform and Version	amd64
CentOS/RHEL 6	✅
CentOS/RHEL 7	✅
CentOS/RHEL 8	✅
Ubuntu 14.04	✅
Ubuntu 16.04	✅

Ubuntu 18.04	✅
Ubuntu 18.10	✅
Ubuntu 19.04	✅
Debian 8	✅
Debian 9	✅
Debian 10	✅

Sensu agent

Platform and Version	`amd64`	`386`
CentOS/RHEL 6	✅	
CentOS/RHEL 7	✅	
CentOS/RHEL 8	✅	
Ubuntu 14.04	✅	
Ubuntu 16.04	✅	
Ubuntu 18.04	✅	
Ubuntu 18.10	✅	
Ubuntu 19.04	✅	
Debian 8	✅	
Debian 9	✅	
Debian 10	✅	
Windows Server 2008 R2 and later	✅	✅

| Windows 7 and later | ✅ | ✅ |

Sensuctl command line tool

Platform and Version	`amd64`	`386`
CentOS/RHEL 6	✅	
CentOS/RHEL 7	✅	
CentOS/RHEL 8	✅	
Ubuntu 14.04	✅	
Ubuntu 16.04	✅	
Ubuntu 18.04	✅	
Ubuntu 18.10	✅	
Ubuntu 19.04	✅	
Debian 8	✅	
Debian 9	✅	
Debian 10	✅	

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 |

Integrations

- ¬ [Sensu Go Data Source plugin for Grafana](#)
- ¬ [Chef cookbook](#)
- ¬ [Puppet module](#)
- ¬ [Ansible role](#)

Binary-only distributions

[Binary-only distributions](#) that contain the Sensu backend, agent, and sensuctl tool are available in `.zip` and `.tar.gz` formats.

Platform & Version	`amd64`	`arm64`	`armv5`	`armv6`	`armv7`	`386`
Linux	✅	✅	✅	✅	✅	✅
Windows	✅					✅
macOS	✅					
Solaris	✅					

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 homepage. See the [feature comparison matrix](#) to learn more. To build Sensu Go from source, see the [contributing guide on GitHub](#)

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.Sensu checks use the same specification as **Nagios**, so you can use Nagios check plugins with Sensu.

You can use checks to monitor server resources, services, and application health (for example, to check whether Nginx is running) and collect and analyze metrics (for example, to learn how much disk space you have left).

Use checks to monitor a service

This guide will help you monitor server resources (specifically, CPU usage) by configuring a check named `check-cpu` with a subscription named `system` to target all entities that are subscribed to the `system` subscription.To use this guide, you'll need to install a Sensu backend and have at least one Sensu agent running on Linux.

Register assets

To power the check, you'll use the <u>Sensu CPU Checks</u> asset and the <u>Sensu Ruby Runtime</u> asset.

Use `sensuctl asset add` to register the `sensu-plugins-cpu-checks` asset:

```
sensuctl asset add sensu-plugins/sensu-plugins-cpu-checks:4.1.0 -r cpu-checks-
plugins
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `cpu-checks-plugins`.

You can also download the asset definition for Debian or Alpine from <u>Bonsai</u> and register the asset with `sensuctl create --file filename.yml`.

Then, use the following sensuctl example to register the `sensu-ruby-runtime` asset:

```
sensuctl asset add sensu/sensu-ruby-runtime:0.0.10 -r sensu-ruby-runtime
```

You can also download the asset definition from Bonsai and register the asset using `sensuctl create --file filename.yml`.

Use sensuctl to confirm that both the `cpu-checks-plugins` and `sensu-ruby-runtime` assets are ready to use:

```
sensuctl asset list
     Name                           URL                          Hash
─────────────────────────────

────────────────────────────────────────────────────────────────────────────
──────────
 cpu-checks-plugins  //assets.bonsai.sensu.io/.../sensu-plugins-cpu-checks_4.1.0_centos_linux_amd64.tar.gz     518e7c1
 sensu-ruby-runtime      //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.0.10_ruby-2.4.4_centos_linux_amd64.tar.gz
 338b88b
```

Create a check

Now that the assets are registered, create a check named `check-cpu` that runs the command `check-cpu.rb -w 75 -c 90` with the `cpu-checks-plugins` and `sensu-ruby-runtime` assets 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.rb -w 75 -c 90' \
--interval 60 \
--subscriptions system \
--runtime-assets cpu-checks-plugins,sensu-ruby-runtime
```

Configure the subscription

To run the check, you'll need a Sensu agent with the subscription `system`. After you install an agent, open `/etc/sensu/agent.yml` and add the `system` subscription so the subscription configuration looks like this:

```
subscriptions:
  - system
```

Then, restart the agent:

```
sudo service sensu-agent restart
```

Validate the check

Use sensuctl to confirm that Sensu is monitoring CPU usage using the `check-cpu`, returning an OK status (`0`). 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 Sensu backend.

```
sensuctl event list
   Entity     Check                                          Output                                                Status
 Silenced          Timestamp
 ──────────────── ────────────
 ──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 ────────────────────────────────────────────────────────────── ──────── ───────────
 ────────────────────────────────────
  sensu-centos   check-cpu   CheckCPU TOTAL OK:  total=0.2 user=0.0 nice=0.0 system=0.2 idle=99.8 iowait=0.0 irq=0.0
 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0      0   false     2019-04-23 16:42:28 +0000 UTC
```

Next steps

Now that you know how to run a check to monitor CPU usage, read these resources to learn more:

- ¬ Checks reference

- ¬ Install plugins with assets

Monitor external resources with proxy requests and 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` attributes, Sensu agents report the resulting event under the proxy entity instead of the agent entity.

NOTE: *This guide requires a running Sensu backend, a running Sensu agent, and a sensuctl instance configured to connect to the backend as a user with get, list, and create permissions for entities, checks, and events.*

Use a proxy entity to monitor a website

In this section, you'll monitor the status of sensu.io by configuring a check with a **proxy entity name** so that Sensu creates an entity that represents the site and reports the status of the site under this entity.

Register assets

To power the check, use the Sensu Plugins HTTP asset and the Sensu Ruby Runtime asset.

Use `sensuctl asset add` to register the `sensu-plugins-http` asset:

```
sensuctl asset add sensu-plugins/sensu-plugins-http:5.1.1 -r sensu-plugins-http
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `sensu-plugins-http`.

You can also download the asset definition for Debian or Alpine from <u>Bonsai</u> and register the asset with `sensuctl create --file filename.yml` .

Then, use the following sensuctl example to register the `sensu-ruby-runtime` asset:

```
sensuctl asset add sensu/sensu-ruby-runtime:0.0.10 -r sensu-ruby-runtime
```

You can also download the asset definition from <u>Bonsai</u> and register the asset using `sensuctl create --file filename.yml` .

Use sensuctl to confirm that both the `sensu-plugins-http` and `sensu-ruby-runtime` assets are ready to use:

```
sensuctl asset list
      Name                          URL                    Hash
 ————————————————————————————
 ——————————————————————————————————————————————————————————
 —————————
  sensu-plugins-http      //assets.bonsai.sensu.io/.../sensu-plugins-http_5.1.1_centos_linux_amd64.tar.gz      31023af
  sensu-ruby-runtime       //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.0.10_ruby-2.4.4_centos_linux_amd64.tar.gz
  338b88b
```

Create the check

Now that the assets are registered, you can create a check named `check-sensu-site` to run the command `check-http.rb -u https://sensu.io` with the `sensu-plugins-http` and `sensu-ruby-runtime` assets, at an interval of 60 seconds, for all agents subscribed to the `proxy` subscription, using the `sensu-site` proxy entity name.To avoid duplicate events, add the `round_robin` attribute to distribute the check execution across all agents subscribed to the `proxy` subscription.

Create a file called `check.json` and add this check definition:

YML

```
type: CheckConfig
api_version: core/v2
metadata:
```

```
    name: check-sensu-site
    namespace: default
  spec:
    command: check-http.rb -u https://sensu.io
    interval: 60
    proxy_entity_name: sensu-site
    publish: true
    round_robin: true
    runtime_assets:
    - sensu-plugins-http
    - sensu-ruby-runtime
    subscriptions:
    - proxy
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-sensu-site",
    "namespace": "default"
  },
  "spec": {
    "command": "check-http.rb -u https://sensu.io",
    "runtime_assets": [
      "sensu-plugins-http",
      "sensu-ruby-runtime"
    ],
    "interval": 60,
    "proxy_entity_name": "sensu-site",
    "publish": true,
    "round_robin": true,
    "subscriptions": [
      "proxy"
    ]
  }
}
```

Now you can use sensuctl to add the check to Sensu:

```
sensuctl create --file check.json
```

```
sensuctl check list
     Name              Command                Interval  Cron  Timeout  TTL  Subscriptions  Handlers          Assets
 Hooks  Publish?  Stdin?
 ─────────────────── ───────────────────────────────── ────────── ─────── ─────────── ─────
 ─────────────── ─────────── ───────────────────────────────────────── ──────── ──────────
 ──────────

 check-sensu-site  check-http.rb -u https://sensu.io     60           0   0  proxy          sensu-plugins-http,sensu-
 ruby-runtime        true    false
```

Add the subscription

To run the check, you'll need a Sensu agent with the subscription `proxy` .After you install an agent,
open `/etc/sensu/agent.yml` and add the `proxy` subscription so the subscription configuration
looks like this:

```
subscriptions:
  - proxy
```

Then, restart the agent:

```
sudo service sensu-agent restart
```

Validate the check

Use sensuctl to confirm that Sensu created the proxy entity `sensu-site` :

```
sensuctl entity list
     ID      Class   OS        Subscriptions           Last Seen
 ─────────────── ─────── ─────── ───────────────────────────
 ──────────────────────────────────

 sensu-centos  agent  linux  proxy,entity:sensu-centos  2019-01-16 21:50:03 +0000 UTC
```

| sensu-site | proxy | entity:sensu-site | N/A |

NOTE: *It might take a few moments for Sensu to execute the check and create the proxy entity.*

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
=== sensu-site - check-sensu-site
Entity:    sensu-site
Check:     check-sensu-site
Output:
Status:    0
History:   0,0
Silenced:  false
Timestamp: 2019-01-16 21:51:53 +0000 UTC
```

You can also see the new proxy entity in your Sensu dashboard.

Use proxy requests to monitor a group of websites

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 the `proxy_requests` check attribute along with entity labels and token substitution to monitor three sites with the same check. Before you start, register the `sensu-plugins-http` and `sensu-ruby-runtime` assets 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.

Create a file called `entities.json` and add the following entity definitions:

```
{
```

```json
    "type": "Entity",
    "api_version": "core/v2",
    "metadata": {
      "name": "sensu-docs",
      "namespace": "default",
      "labels": {
        "proxy_type": "website",
        "url": "https://docs.sensu.io"
      }
    },
    "spec": {
      "entity_class": "proxy"
    }
  }
  {
    "type": "Entity",
    "api_version": "core/v2",
    "metadata": {
      "name": "packagecloud-site",
      "namespace": "default",
      "labels": {
        "proxy_type": "website",
        "url": "https://packagecloud.io"
      }
    },
    "spec": {
      "entity_class": "proxy"
    }
  }
  {
    "type": "Entity",
    "api_version": "core/v2",
    "metadata": {
      "name": "github-site",
      "namespace": "default",
      "labels": {
        "proxy_type": "website",
        "url": "https://github.com"
      }
    },
    "spec": {
      "entity_class": "proxy"
```

```
        }
    }
}
```

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.*

Now you can use sensuctl to add these proxy entities to Sensu:

```
sensuctl create --file entities.json

sensuctl entity list
      ID        Class  OS      Subscriptions            Last Seen
 _____ _____ _____ _____
 _____
 github-site     proxy                 N/A
 packagecloud-site  proxy               N/A
 sensu-centos      agent  linux  proxy,entity:sensu-centos  2019-01-16 23:05:03 +0000 UTC
 sensu-docs      proxy                 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.

Create a file called `check-proxy-requests.json` and add the following check definition:

YML

```
type: CheckConfig
api_version: core/v2
metadata:
  name: check-http
  namespace: default
spec:
  command: check-http.rb -u {{ .labels.url }}
  interval: 60
  proxy_requests:
    entity_attributes:
```

```
      - entity.entity_class == 'proxy'
      - entity.labels.proxy_type == 'website'
  publish: true
  runtime_assets:
  - sensu-plugins-http
  - sensu-ruby-runtime
  subscriptions:
  - proxy
```

JSON

```json
  {
    "type": "CheckConfig",
    "api_version": "core/v2",
    "metadata": {
      "name": "check-http",
      "namespace": "default"
    },
    "spec": {
      "command": "check-http.rb -u {{ .labels.url }}",
      "runtime_assets": [
        "sensu-plugins-http",
        "sensu-ruby-runtime"
      ],
      "interval": 60,
      "subscriptions": [
        "proxy"
      ],
      "publish": true,
      "proxy_requests": {
        "entity_attributes": [
          "entity.entity_class == 'proxy'",
          "entity.labels.proxy_type == 'website'"
        ]
      }
    }
  }
```

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, you can use token substitution to apply the correct `url` in the check `command` .

Use sensuctl to add the `check-proxy-requests` check to Sensu:

```
sensuctl create --file check-proxy-requests.json

sensuctl check list
     Name               Command              Interval  Cron  Timeout  TTL  Subscriptions  Handlers         Assets
Hooks  Publish?  Stdin?
 ──────────────────── ───────────────────────────────────────────── ────────── ─────── ────────── ─────
 ──────────────── ────────── ────────────────────────────────────────── ─────── ──────────
 ─────────
  check-http      check-http.rb -u {{ .labels.url }}      60         0   0  proxy          sensu-plugins-http,sensu-ruby-
runtime       true     false
```

PRO TIP: *To distribute check executions across multiple agents, set the* `round-robin` *check attribute to* `true` *. For more information about round robin checks, see the* *check reference.*

Validate the check

Before you validate the check, make sure that you've registered the `sensu-plugins-http` and `sensu-ruby-runtime` assets and added the `proxy` 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` , returning a status of `0` (OK):

```
sensuctl event list
     Entity            Check          Output  Status  Silenced        Timestamp
 ──────────────────── ──────────────────────── ──────── ──────── ──────────
 ────────────────────────────────────────
 github-site       check-http                 0  false    2019-01-17 17:10:31 +0000 UTC
 packagecloud-site  check-http                  0  false    2019-01-17 17:10:34 +0000 UTC
 sensu-centos       keepalive          ...    0  false    2019-01-17 17:10:34 +0000 UTC
 sensu-docs         check-http                 0  false    2019-01-17 17:06:59 +0000 UTC
```

Next steps

Now that you know how to run a proxy check to verify the status of a website and use proxy requests to run a check on two different proxy entities based on label evaluation, read these recommended resources:

- ¬ [Proxy checks](#)

- ¬ [Assets reference](#)

- ¬ [Send Slack alerts with handlers](#)

Collect metrics with Sensu checks

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 or want to learn how to configure a check before reading this guide, read the [check reference](#) and [Monitor server resources](#).

Extract metrics from check output

To extract metrics from check output, you'll need to:

1. Configure the check `command` so that the command execution outputs metrics in one of the [supported output metric formats](#).
2. Configure the check `output_metric_format` to one of the [supported output metric formats](#).

You can also configure the check `output_metric_handlers` to a Sensu handler that is equipped to handle Sensu metrics if you wish. See [handlers](#) or [influx-db handler](#) to learn more.

You can configure the check with these fields at creation or use the commands in this guide (assuming you have a check named `collect-metrics`). This example uses `graphite_plaintext` format and sends the metrics to a handler named `influx-db` .

```
sensuctl check set-command collect-metrics collect_metrics.sh
sensuctl check set-output-metric-format collect-metrics graphite_plaintext
sensuctl check set-output-metric-handlers collect-metrics influx-db
```

Supported output metric formats

The output metric formats that Sensu currently supports for check output metric extraction are `nagios` , `influxdb` , `graphite` , and `opentsdb` .

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
```

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
```

opentsdb

output_metric_format	`opentsdb_line`
documentation	[OpenTSDB Data Specification](#)
example	

```
        sys.cpu.user 1356998400 42.5 host=webserver01 cpu=0
```

Validate the 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 event pipeline. You should expect to see errors logged by sensu-agent if it is unable to parse the check output. To confirm that metrics have been extracted from your check, inspect the event passed to the handler.

See Troubleshooting for an example debug handler that writes events to a file for inspection.

The example check would yield an event similar to this:

YML

```
type: Event
api_version: core/v2
metadata: {}
spec:
  check:
    command: collect_metrics.sh
    metadata:
      name: collect-metrics
      namespace: default
    output: |-
      cpu.idle_percentage 61 1525462242
      mem.sys 104448 1525462242
    output_metric_format: graphite_plaintext
    output_metric_handlers:
    - influx-db
  metrics:
    handlers:
    - influx-db
    points:
    - name: cpu.idle_percentage
      tags: []
      timestamp: 1525462242
      value: 61
    - name: mem.sys
      tags: []
```

```
        timestamp: 1525462242
        value: 104448
```

JSON

```json
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "check": {
      "metadata": {
        "name": "collect-metrics",
        "namespace": "default"
      },
      "command": "collect_metrics.sh",
      "output": "cpu.idle_percentage 61 1525462242\nmem.sys 104448 1525462242",
      "output_metric_format": "graphite_plaintext",
      "output_metric_handlers": [
        "influx-db"
      ]
    },
    "metrics": {
      "handlers": [
        "influx-db"
      ],
      "points": [
        {
          "name": "cpu.idle_percentage",
          "value": 61,
          "timestamp": 1525462242,
          "tags": []
        },
        {
          "name": "mem.sys",
          "value": 104448,
          "timestamp": 1525462242,
          "tags": []
        }
      ]
    }
```

```
    }
  }
```

Next steps

Now you know how to extract metrics from check output!Check out these resources for more information about scheduling checks and using handlers:

- ¬ Checks reference: in-depth checks documentation

- ¬ Monitor server resources: learn how to schedule checks

- ¬ Handlers reference: in-depth handler documentation

- ¬ Populate metrics in InfluxDB: learn to use Sensu's built-in metrics handler

Augment event data with check hooks

Check hooks are **commands** the Sensu agent runs in response to the result of **check** command execution. The Sensu agent executes the appropriate configured hook command based on the exit status code of the check command (e.g. `1`).

Check hooks allow Sensu users to automate data collection that operators would routinely perform to investigate monitoring alerts, which frees up precious operator time. Although you can use check hooks for rudimentary auto-remediation tasks, they are intended to enrich monitoring event data. This guide helps you create a check hook that captures the process tree in case a service check returns a critical status.

Use check hooks to gather context

Follow these steps to create a check hook that captures the process tree in the event that an `nginx_process` check returns a status of `2` (critical, not running).

1. Create a hook

Create a new hook that runs a specific command to capture the process tree. Set an execution **timeout** of 10 seconds for this command:

```
sensuctl hook create process_tree  \
--command 'ps aux' \
--timeout 10
```

2. Assign the hook to a check

Now that you've created the `process_tree` hook, you can assign it to a check. This example assumes you've already set up the `nginx_process` 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:

```
sensuctl check set-hooks nginx_process  \
--type critical \
--hooks process_tree
```

3. Validate the check hook

Verify that the check hook is behaving properly against a specific event with `sensuctl`. It might take a few moments after you assign the check hook for the check to be scheduled on the entity and the result sent back to the Sensu backend. The check hook command result is available in the `hooks` array, within the `check` scope:

```
sensuctl event info i-424242 nginx_process --format json

{
  [...]
  "check": {
    [...]
    "hooks": [
      {
        "config": {
          "name": "process_tree",
          "command": "ps aux",
          "timeout": 10,
          "namespace": "default"
        },
        "duration": 0.008713605,
        "executed": 1521724622,
        "output": "",
        "status": 0
      }
    ],
    [...]
  }
}
```

After you confirm that the hook is attached to your check, you can stop Nginx and observe the check hook in action on the next check execution. This example uses sensuctl to query event info and send the response to `jq` so you can isolate the check hook output:

```
sensuctl event info i-424242 nginx_process --format json | jq -r
'.check.hooks[0].output'

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]
```

Although this output is truncated in the interest of brevity, it reflects the output of the `ps aux` command specified in the check hook you created. 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.

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).

NOTE: On Windows machines running Sensu, the StatsD UDP port is not supported. Instead, the TCP port is exposed.

Configure the StatsD listener

Use 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

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.

To configure journald to forward logging data to syslog, modify `/etc/systemd/journald.conf` to include the following line:

```
ForwardToSyslog=yes
```

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.

```
# For the sensu-backend service, inside /etc/rsyslog.d/99-sensu-backend.conf
if $programname == 'sensu-backend' then {
        /var/log/sensu/sensu-backend.log

        ~
}

# For the sensu-agent service, inside /etc/rsyslog.d/99-sensu-agent.conf
if $programname == 'sensu-agent' then {
        /var/log/sensu/sensu-agent.log

        ~
}
```

Restart rsyslog and journald to apply the new configuration:

```
systemctl restart systemd-journald
systemctl restart rsyslog
```

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.

```
# Inside /etc/logrotate.d/sensu-agent.conf
/var/log/sensu/sensu-agent.log {
    daily
    rotate 7
    size 100M
    compress
    delaycompress
    postrotate
      /bin/systemctl restart rsyslog
    endscript
}

# Inside /etc/logrotate.d/sensu-backend.conf
/var/log/sensu/sensu-backend.log {
    daily
    rotate 7
    size 100M
    compress
    delaycompress
    postrotate
      /bin/systemctl restart rsyslog
    endscript
}
```

You can use the following command to see 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
```

NOTE: *On Ubuntu systems, be sure to run* `chown -R syslog:adm /var/log/sensu` *so syslog can*

write to that directory.

Populate metrics in InfluxDB with handlers

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](#). If you're not familiar with handlers, consider reading the [handlers reference](#) before continuing through this guide.

Use a handler to populate InfluxDB

The example in this guide explains how to populate Sensu metrics into the time series database [InfluxDB](#). Metrics can be collected from [check output](#) or the [Sensu StatsD Server](#).

Register the asset

[Assets](#) are shareable, reusable packages that make it easier to deploy Sensu plugins. This example uses the [Sensu InfluxDB Handler](#) asset to power an `influx-db` handler.

Use `sensuctl asset add` to register the [Sensu InfluxDB Handler](#) asset:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.1.2 -r influxdb-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `influxdb-handler`.

You can also download the latest asset definition for your platform from [Bonsai](#) and register the asset with `sensuctl create --file filename.yml`.

You should see a confirmation message from sensuctl:

```
Created
```

Run `sensuctl asset list --format yaml` to confirm that the asset is ready to use.

Create the handler

Now that you have registered the asset, you'll use sensuctl to create a handler called `influx-db` that pipes event data to InfluxDB with the `sensu-influxdb-handler` asset. Edit the command below to include your database name, address, username, and password. For more information about the Sensu InfluxDB handler, see the asset page in Bonsai.

```
sensuctl handler create influx-db \
--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 influxdb-handler
```

You should see a confirmation message from sensuctl:

```
Created
```

Assign the handler to an event

With the `influx-db` handler created, you can assign it to a check for check output metric extraction. In this example, the check name is `collect-metrics`:

```
sensuctl check set-output-metric-handlers collect-metrics influx-db
```

You can also assign the handler to the Sensu StatsD listener at agent startup to pass all StatsD metrics into InfluxDB:

```
sensu-agent start --statsd-event-handlers influx-db
```

Validate the handler

It might take a few moments after you assign the handler to the check or StatsD server for Sensu to receive the metrics, but after an event is handled you should start to see metrics populating InfluxDB. You can verify proper handler behavior with `sensu-backend` logs. See Troubleshooting for log locations by platform.

Whenever an event is being handled, a log entry is added with the message `"handler":"influx-db","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 a handler to metrics and take action on events, here are a few other recommended resources:

- Handlers reference
- Aggregate metrics with the Sensu StatsD listener
- Collect metrics with Sensu checks

Send Slack alerts with handlers

Sensu event handlers are actions the Sensu backend executes on events. You can use handlers to send an email alert, create or resolve incidents (in PagerDuty, for example), or store metrics in a time-series database like InfluxDB.

Use a handler to send alerts to Slack

This guide will help you send alerts to Slack in the channel `monitoring` by configuring a handler named `slack` to a check named `check-cpu`. If you don't already have a check in place, Monitor server resources is a great place to start.

Register the asset

Assets are shareable, reusable packages that help you deploy Sensu plugins. In this guide, you'll use the Sensu Slack Handler asset to power a `slack` handler.

Use `sensuctl asset add` to register the Sensu Slack Handler asset:

```
sensuctl asset add sensu/sensu-slack-handler:1.0.3 -r sensu-slack-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `sensu-slack-handler`.

You can also download the latest asset definition for your platform from Bonsai and register the asset with `sensuctl create --file filename.yml`.

You should see a confirmation message from sensuctl:

```
Created
```

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. After saving, you'll see your webhook URL under Integration Settings.

Create a handler

Use sensuctl to create a handler called `slack` that pipes event data to Slack using the `sensu-slack-handler` asset. Edit the command below to include your Slack channel and webhook URL. For more information about customizing your Sensu slack alerts, see the asset page in Bonsai.

```
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 see a confirmation message from sensuctl:

```
Created
```

Assign the handler to a check

With the `slack` handler created, you can assign it to a check. In this case, you're using the `check-cpu` check: you want to receive Slack alerts whenever the CPU usage of your systems reach some specific thresholds. Assign your handler to the check `check-cpu`:

```
sensuctl check set-handlers check-cpu slack
```

Validate the handler

It might take a few moments after you assign the handler to the check for the check to be scheduled on the entities and the result sent back to Sensu backend. After an event is handled, you should see the following message in Slack:
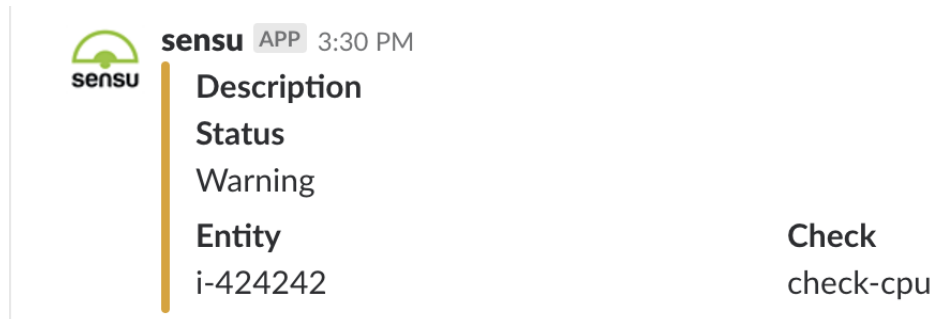
Verify the proper behavior of this handler with `sensu-backend` logs. See [Troubleshooting](#) for log locations by platform.

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":"pipelined executed event pipe handler","output":"","status":0`.

Next steps

Now that you know how to apply a handler to a check and take action on events, read the [handlers reference](#) for in-depth handler documentation and check out the [Reduce alert fatigue](#) guide.

You can also try our interactive tutorial and learn how to [send Sensu Go alerts to your PagerDuty account](#).

Send email alerts with the Sensu Go Email Handler

Sensu event handlers are actions the Sensu backend executes on events. This guide explains how to use the Sensu Go Email Handler asset to send notification emails.

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 an email handler that sends notifications to the email address you specify. You'll also add an event filter to make sure you only receive a notification when your event represents a status change.

Add the email handler asset

Assets are shareable, reusable packages that help you deploy Sensu plugins. In this guide, you'll use the Sensu Go Email Handler asset to power an `email` handler.

Use the following sensuctl example to register the Sensu Go Email Handler asset:

```
sensuctl asset add sensu/sensu-email-handler -r email-handler
```

The -r (rename) flag allows you to specify a shorter name for the asset (in this case, `email-`

`handler`).

You can also download the latest asset definition for your platform from Bonsai and register the asset with `sensuctl create --file filename.yml` .

To confirm that the handler asset was added correctly, run:

```
sensuctl asset list
```

You should see the `email-handler` asset in the list. For a detailed list of everything related to the asset that Sensu added automatically, run:

```
sensuctl asset info email-handler
```

The asset includes the `sensu-email-handler` command, which you will use when you create the email handler definition later in this guide.

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:

```
cat << EOF | sensuctl create
---
type: EventFilter
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: state_change_only
  namespace: default
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 `sensu/sensu-email-handler` asset will send notifications. In the handler definition's `command` value you'll need to change a few things.

Copy this text into a text editor:

```
cat << EOF | sensuctl create
---
api_version: core/v2
type: Handler
metadata:
  namespace: default
  name: email
spec:
  type: pipe
  command: sensu-email-handler -f YOUR-SENDER@example.com -t YOUR-
RECIPIENT@example.com -s YOUR-SMTP-SERVER.example.com
    -u USERNAME -p PASSWORD
  timeout: 10
```

```
    filters:
    - is_incident
    - not_silenced
    - state_change_only
    runtime_assets:
    - email-handler
  EOF
```

Then, replace the following text:

- `YOUR-SENDER@example.com` : Replace with the email address you want to use to send email alerts.

- `YOUR-RECIPIENT@example.com` : Replace with the email address you want to receive email alerts.

- `YOUR-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 for sending email via SMTP.If you have enabled 2-step verification on your Google account, you'll need to 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.

You probably noticed that the handler definition includes two other filters besides `state_change_only` : `is_incident` and `not_silenced` .These two filters are included in every Sensu backend installation, so you don't have to create them.

After you add your email, server, username, and password values, run your updated code to create the email handler definition.

Now your handler and event filter are set up!

The Sensu Go Email Handler 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.

Before your handler 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 credentials 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
efPxbRciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NzkwMzY5NjQsImp0aSI6ImJiMmY0ODY4ZTJ
hZWEyMDhhMTExOTllMGZkNzkzMDc0Iiwic3ViIjoiYWRtaW4iLCJncm91cHMiOlsiY2x1c3Rlci1hZG1pbnM
iLCJzeXN0ZW06dXNlcnMiXSwicHJvdmlkZXIiOnsicHJvdmlkZXJfaWQiOiJiYXNpYyIsInByb3ZpZGVyX3R
5cGUiOiIiLCJ1c2VyX2lkIjoiYWRtaW4ifX0.6XmuvblCN743R2maF4yErS3K3sOVczsCBsjib9TenUU
```

With the environment variables set, you can use the Sensu API to create your ad hoc monitoring event. 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,
    "interval": 60
  }
```

```
}' \
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 see 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,
    "interval": 60,
    "handlers": ["email"]
  }
}' \
http://localhost:8080/api/core/v2/namespaces/default/events/server01/server-health
```

NOTE: *If you see an* `invalid credentials` *error, refresh your token.Run* `eval $(sensuctl env)` *.*

Check your email — you should see 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,
      "interval": 60,
      "handlers": ["email"]
    }
  }' \
  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.

- ⌐ Read the handlers reference for in-depth handler documentation.

- ⌐ Check out the Reduce alert fatigue guide.

You can also follow our Up and running with Sensu Go interactive tutorial to set up the Sensu Go email handler and test a similar workflow with the addition of a Sensu agent for producing events using scheduled checks.

Install plugins with assets

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. See the asset reference for more information about assets. This guide uses the Sensu PagerDuty Handler asset as an example.

1. Register the Sensu PagerDuty Handler asset

To add the Sensu PagerDuty Handler asset to Sensu, use `sensuctl asset add` :

```
sensuctl asset add sensu/sensu-pagerduty-handler:1.2.0 -r pagerduty-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `pagerduty-handler` .

You can also click the Download button on the asset page in Bonsai to download the asset definition for your Sensu backend platform and architecture.

2. 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 1.2.0 of the Sensu PagerDuty Handler asset for Linux AMD64:

```
---
type: Asset
api_version: core/v2
metadata:
  name: pagerduty-handler
  namespace: default
  labels: {}
  annotations: {}
spec:
  url:
https://assets.bonsai.sensu.io/02fc48fb7cbfd27f36915489af2725034a046772/sensu-
pagerduty-handler_1.2.0_linux_amd64.tar.gz
  sha512:
5be236b5b9ccceb10920d3a171ada4ac4f4caaf87f822475cd48bd7f2fab3235fa298f79ef6f97b0eb64
98205740bb1af1120ca036fd3381edfebd9fb15aaa99
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
```

Filters for *check* assets should match entity platforms. Filters for *handler and filter* 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 assets (like the ServiceNow and Jira event handlers) require a Sensu commercial license. For more information about commercial features and to activate your license, see Get started with commercial features.

Use sensuctl to verify that the asset is registered and ready to use:

```
sensuctl asset list --format yaml
```

3. Create a workflow

With the asset downloaded and registered, you can use it in a monitoring workflow. 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 <u>Sensu PagerDuty Handler</u> 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

```yaml
type: Handler
api_version: core/v2
metadata:
  name: pagerduty
  namespace: default
spec:
  command: sensu-pagerduty-handler
  env_vars:
  - PAGERDUTY_TOKEN=SECRET
  filters:
  - is_incident
  runtime_assets:
  - pagerduty-handler
  timeout: 10
  type: pipe
```

JSON

```json
{
    "api_version": "core/v2",
    "type": "Handler",
    "metadata": {
        "namespace": "default",
        "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.json`), and add it to Sensu with sensuctl:

```
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.See Monitor server resources to learn more.

Next steps

Read these resources for more information about using assets in Sensu:

- Assets reference
- Asset format specification
- Share assets on Bonsai

You can also try our interactive tutorial to send critical alerts to your PagerDuty account.

Reduce alert fatigue with filters

Sensu event filters allow you to filter events destined for one or more event handlers.Sensu event filters evaluate their expressions against the event data to determine whether the event should be passed to an event handler.

Use event filters to eliminate notification noise from recurring events and to filter events from systems in pre-production environments.

Use event filters to reduce alert fatigue

In this guide, you learn how to reduce alert fatigue by configuring an event filter named `hourly` for a handler named `slack` to prevent alerts from being sent to Slack every minute.If you don't already have a handler in place, follow Send Slack alerts with handlers before continuing with this guide.

You can use either of two approaches to create the event filter to handle occurrences:

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"
```

Assign the event filter to a handler

Now that you've created the `hourly` event filter, you can assign it to a handler.Because you want to reduce the number of Slack messages Sensu sends, you'll apply the event filter to an existing handler named `slack`, in addition to the built-in `is_incident` filter, so only failing events are handled.

```
sensuctl handler update slack
```

Follow the prompts to add the `hourly` and `is_incident` event filters to the Slack handler.

Create a fatigue check event filter

Although you can use `sensuctl` to interactively create a filter, you can create more reusable filters with assets.Read on to see how to implement a filter using this approach.

Approach 2: Use an event filter asset

If you're not already familiar with assets, please take a moment to read Install plugins with assets.This will help you understand what assets are and how they are used in Sensu.

In this approach, the first step is to obtain an event filter 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 fatigue check filter asset:

```
sensuctl asset add nixwiz/sensu-go-fatigue-check-filter:0.3.2 -r fatigue-filter
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `fatigue-filter`.

You can also download the asset directly from Bonsai, the Sensu asset index.

You've registered the asset, but you still need to create the filter.To do this, use the following

configuration:

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: fatigue_check
  namespace: default
spec:
  action: allow
  expressions:
  - fatigue_check(event)
  runtime_assets:
  - fatigue-filter
```

Then, create the filter, naming it `sensu-fatigue-check-filter.yml`:

```
sensuctl create -f sensu-fatigue-check-filter.yml
```

Now that you've created the filter asset and the event filter, you can create the check annotations you need for the asset to work properly.

Annotate a check for filter asset use

Next, you need to make some additions to any checks you want to use the filter with. Here's an example CPU check:

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: linux-cpu-check
  namespace: default
  annotations:
    fatigue_check/occurrences: '1'
    fatigue_check/interval: '3600'
    fatigue_check/allow_resolution: 'false'
```

```
spec:
  command: check-cpu -w 90 c 95
  env_vars:
  handlers:
  - email
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ''
  output_metric_handlers:
  proxy_entity_name: ''
  publish: true
  round_robin: false
  runtime_assets:
  stdin: false
  subdue:
  subscriptions:
  - linux
  timeout: 0
  ttl: 0
```

Notice the annotations under the `metadata` scope.The annotations are required for the filter 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, see the filter asset's README.Next, you'll assign the newly minted event filter to a handler.

Assign the event filter to a handler

Just like with the interactively created event filter, you'll introduce the filter into your Sensu workflow by configuring a handler to use it.Here's an example:

```
---
api_version: core/v2
type: Handler
metadata:
  namespace: default
  name: slack
spec:
  type: pipe
  command: 'sensu-slack-handler --channel ''#general'' --timeout 20 --username
''sensu'' '
  env_vars:
  - SLACK_WEBHOOK_URL=https://www.webhook-url-for-slack.com
  timeout: 30
  filters:
  - is_incident
  - fatigue_check
```

Validate the event filter

Verify the proper behavior of these event filters with `sensu-backend` logs. The default location of these logs varies based on the platform used (see Troubleshooting for details).

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":"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 apply an event filter to a handler and use a filter asset to help reduce alert fatigue, read the filters reference for in-depth information about event filters.

Route alerts with 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, reducing mean time to response and recovery.

In this guide, you'll set up alerts for two teams (ops and dev) with separate Slack channels. Assume each team wants to be alerted only for the things they care about, using their team's Slack channel. To achieve this, you'll create two types of Sensu resources:

- **Event handlers** to store contact preferences for the ops team, the dev team, and a fallback option

- **Event filters** to match contact labels to the right handler

Here's a quick overview of the configuration to set up contact routing. The check definition includes the `contacts: dev` label, which will result in an alert sent to the dev team but not to the ops team or the fallback option.
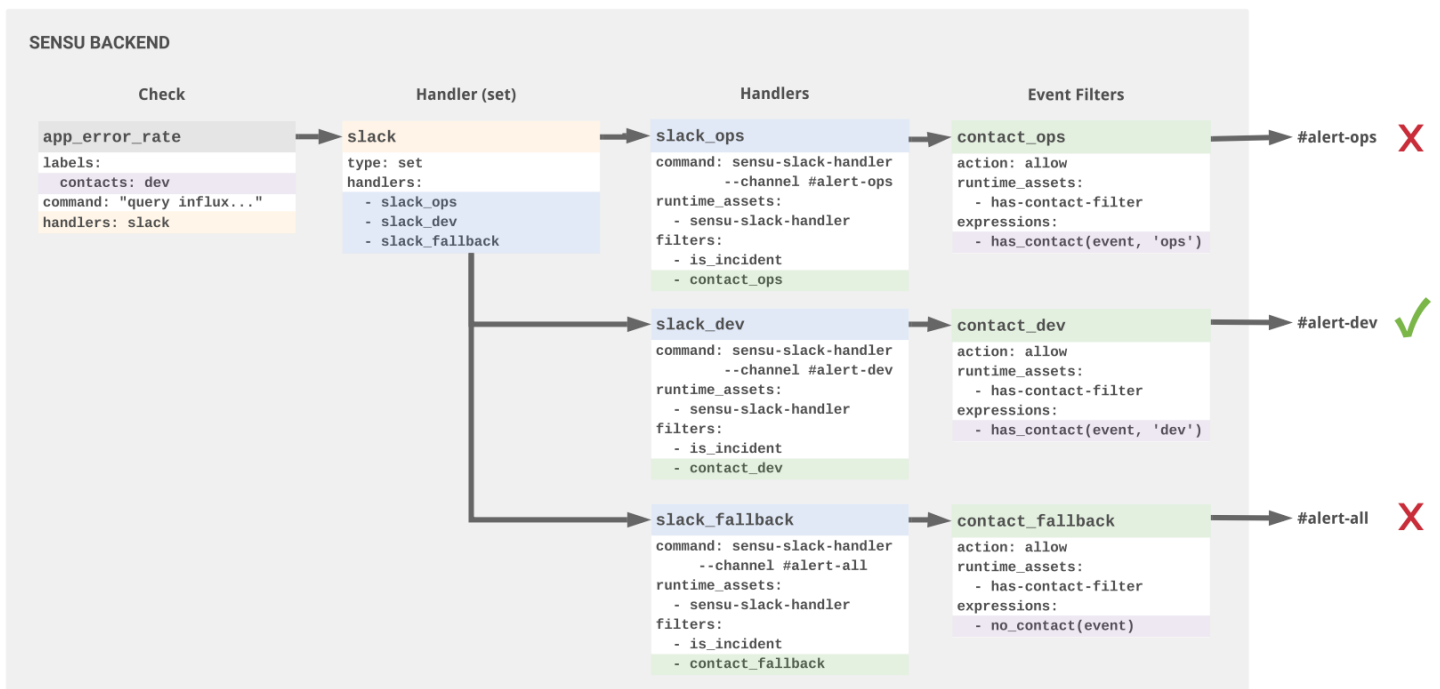
Sensu Go contact routing: Route alerts to the dev team using a check label

Prerequisites

To complete this guide, you'll need:

- A Sensu backend

- At least one Sensu agent

- Sensuctl (configured to talk to the Sensu backend)

- cURL

- A Slack webhook URL and three different Slack channels to receive test alerts (one for each team)

To set up a quick testing environment, download and start the Sensu sandbox.

Configure contact routing

1. Register the has-contact filter asset

Contact routing is powered by the has-contact filter asset. To add the has-contact asset to Sensu, use `sensuctl asset add` :

```
sensuctl asset add sensu/sensu-go-has-contact-filter:0.2.0 -r contact-filter
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `contact-filter` .

You can also download the latest asset definition from Bonsai.

Run `sensuctl asset list --format yaml` to confirm that the asset is ready to use.

2. Create contact filters

The Bonsai documentation for the asset explains that the has-contact 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

To add these filters to Sensu, use `sensuctl create` :

```
echo '---
```

```yaml
type: EventFilter
api_version: core/v2
metadata:
  name: contact_ops
spec:
  action: allow
  runtime_assets:
    - sensu-go-has-contact-filter_any_noarch
  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
```

Run `sensuctl filter list --format yaml` to confirm that the filters are ready to use.

3. Create a handler for each contact

With your contact filters in place, you can create a handler for each contact: ops, dev, and fallback. If you haven't already, add the Slack handler asset to Sensu with sensuctl:

```
sensuctl asset add sensu/sensu-slack-handler:1.0.3 -r sensu-slack-handler
```

This example uses the `-r` (rename) flag to specify a shorter name for the asset: `sensu-slack-handler` .

In each handler definition, specify:

¬ A unique name: `slack_ops` , `slack_dev` , or `slack_fallback`

¬ A customized command with the contact's preferred Slack channel

¬ The contact filter

¬ The built-in `is_incident` and `not_silenced` filters to reduce noise and enable silences

¬ An environment variable that contains your Slack webhook URL

¬ The `sensu-slack-handler` runtime asset

To create the `slack_ops` , `slack_dev` , and `slack_fallback` handlers, edit and run this example:

```
# Edit before running:
# 1. Add your SLACK_WEBHOOK_URL
# 2. Make sure the Slack channels specified in the
#    command` attributes match channels available
#    to receive test alerts in your Slack instance.
echo '---
type: Handler
api_version: core/v2
metadata:
  name: slack_ops
spec:
  command: sensu-slack-handler --channel "#alert-ops"
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters:
  - is_incident
  - not_silenced
  - contact_ops
  runtime_assets:
  - sensu-slack-handler
  type: pipe
```

```
---
type: Handler
api_version: core/v2
metadata:
  name: slack_dev
spec:
  command: sensu-slack-handler --channel "#alert-dev"
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters:
  - is_incident
  - not_silenced
  - contact_dev
  runtime_assets:
  - sensu-slack-handler
  type: pipe
---
type: Handler
api_version: core/v2
metadata:
  name: slack_fallback
spec:
  command: sensu-slack-handler --channel "#alert-all"
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters:
  - is_incident
  - not_silenced
  - contact_fallback
  runtime_assets:
  - sensu-slack-handler
  type: pipe' | sensuctl create
```

Run `sensuctl handler list --format yaml` to confirm that the handlers are ready to use.

4. Create a handler set

To centralize contact management and simplify configuration, create a handler set that combines your contact-specific handlers under a single handler name.

Use `sensuctl` to create a `slack` handler set:

```
echo '---
type: Handler
api_version: core/v2
metadata:
  name: slack
  namespace: default
spec:
  handlers:
  - slack_ops
  - slack_dev
  - slack_fallback
  type: set' | sensuctl create
```

You should see updated output of `sensuctl handler list` that includes the `slack` handler set.

Congratulations! Your Sensu contact routing is set up.Next, test your contact filters to make sure they work.

Test contact routing

To make sure your contact filters work the way you expect, 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"
    },
    "status": 1,
    "output": "You should receive this example event in the Slack channel specified
  by your slack_fallback handler.",
```

```
      "handlers": ["slack"]
    }
  }' \
  http://127.0.0.1:3031/events
```

You should see a 202 response from the API.Since this event doesn't include a `contacts` label, you should also see an alert in the Slack channel specified by the `slack_fallback` handler.Behind the scenes, Sensu uses the `contact_fallback` filter to match the event to the `slack_fallback` handler.

Now, create an event with a `contacts` label:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "example-check",
      "labels": {
        "contacts": "dev"
      }
    },
    "status": 1,
    "output": "You should receive this example event in the Slack channel specified
by your slack_dev handler.",
    "handlers": ["slack"]
  }
}' \
http://127.0.0.1:3031/events
```

Because this event contains the `contacts: dev` label, you should see an alert in the Slack channel specified by the `slack_dev` handler.

Resolve the events by sending the same API requests with `status` set to `0`.

Manage contact labels in checks and entities

To assign an alert to a contact, add a `contacts` label to the check or entity.

Checks

This check definition includes two contacts (`ops` and `dev`) and the handler `slack` .To set up the `check_cpu` check, see Monitor server resources.

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check_cpu
  labels:
    contacts: ops, dev
spec:
  command: check-cpu.rb -w 75 -c 90
  handlers:
  - slack
  interval: 10
  publish: true
  subscriptions:
  - system
  runtime-assets:
  - sensu-plugins-cpu-checks
  - sensu-ruby-runtime
```

When the `check_cpu` check generates an incident, Sensu filters the event according to the `contact_ops` and `contact_dev` filters, resulting in an alert sent to #alert-ops and #alert-dev.
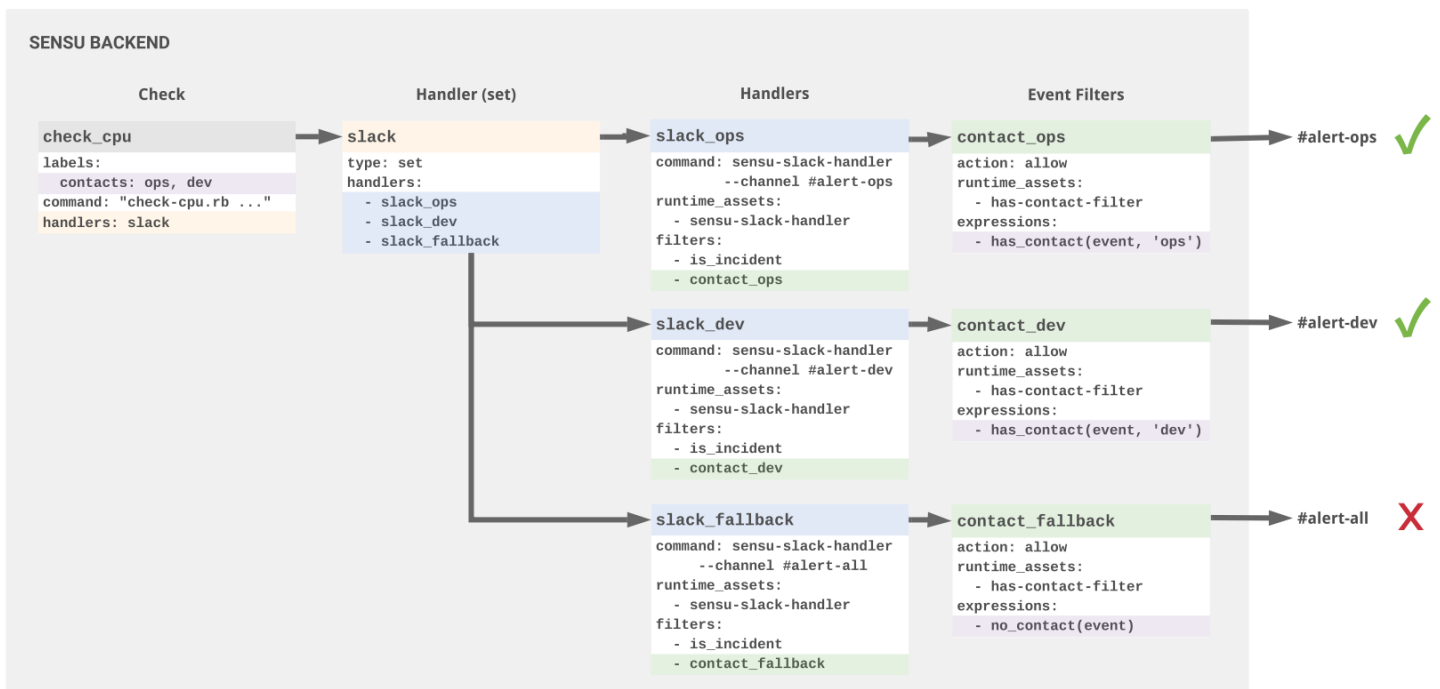
Sensu Go contact routing: Route alerts to two contacts using a check label

Entities

You can also specify contacts using an entity label. For more information about managing entity labels, see the entity 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 #alert-dev but not to #alert-ops or #alert-all.
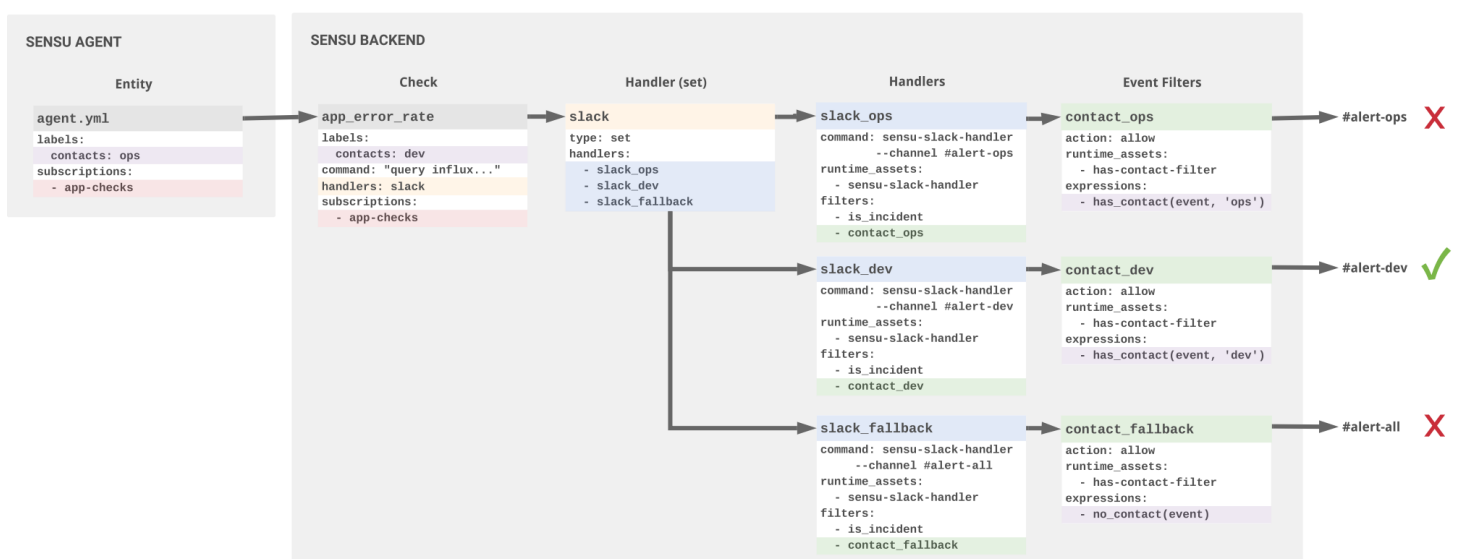
Next steps

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 <u>Reduce alert fatigue</u>.

Deploy Sensu

This guide describes various deployment 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 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. See [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 performance of a Sensu cluster. This means that you should provision Sensu backend infrastructure to provide sustained IO operations per second (IOPS) appropriate for the rate of monitoring events the system will be required to process.

Our hardware requirements documentation describes the minimum and recommended hardware specifications for running the Sensu backend.

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 or subject alternate names must include the network interfaces and DNS names that will point to those systems.

See 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

Single Sensu Go backend or standalone architecture

You can reconfigure a single backend as a member of a cluster, but this operation is destructive: it 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 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.

NOTE: *Multiple Sensu backends can relay their events to a central backend using the sensu-relay-handler.*

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.
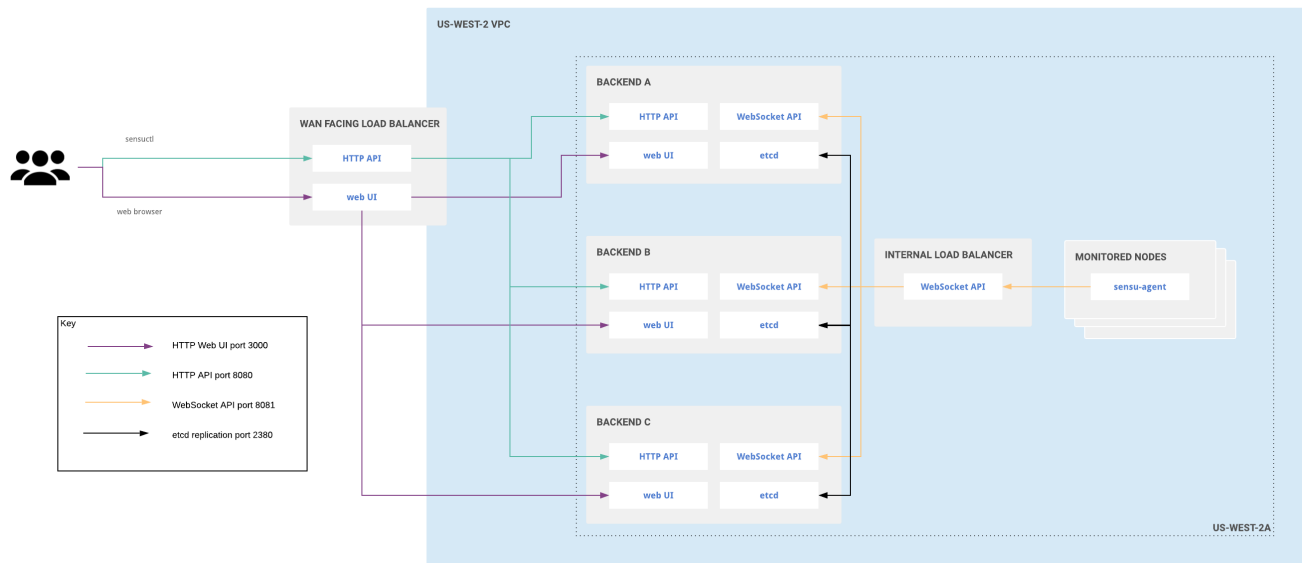
Clustered Sensu Go architecture for a single availability zone

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. See 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.
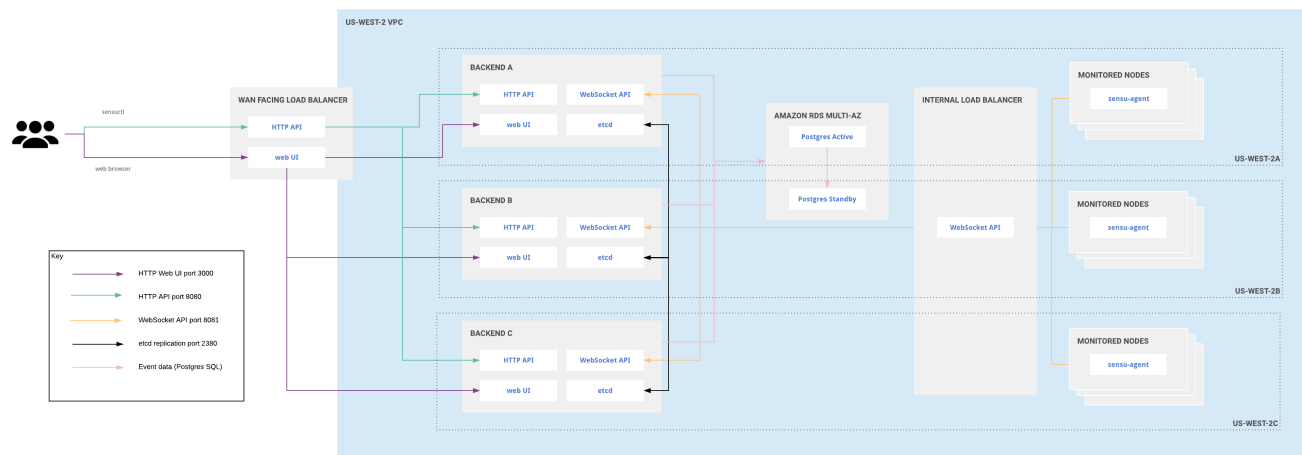
Clustered Sensu Go architecture for multiple availability zones

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.
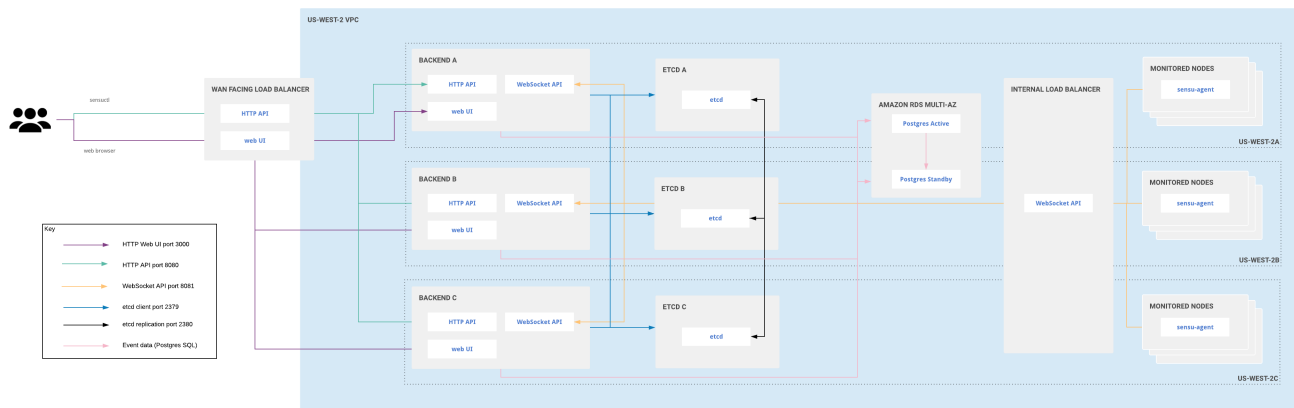
Large-scale clustered Sensu Go architecture for multiple availability zones

Scaled cluster performance with PostgreSQL

To achieve the high rate of event processing that many enterprises require, Sensu supports PostgreSQL event storage as a commmercial feature. See the Datastore reference for details on configuring the Sensu backend to use PostgreSQL for event storage.
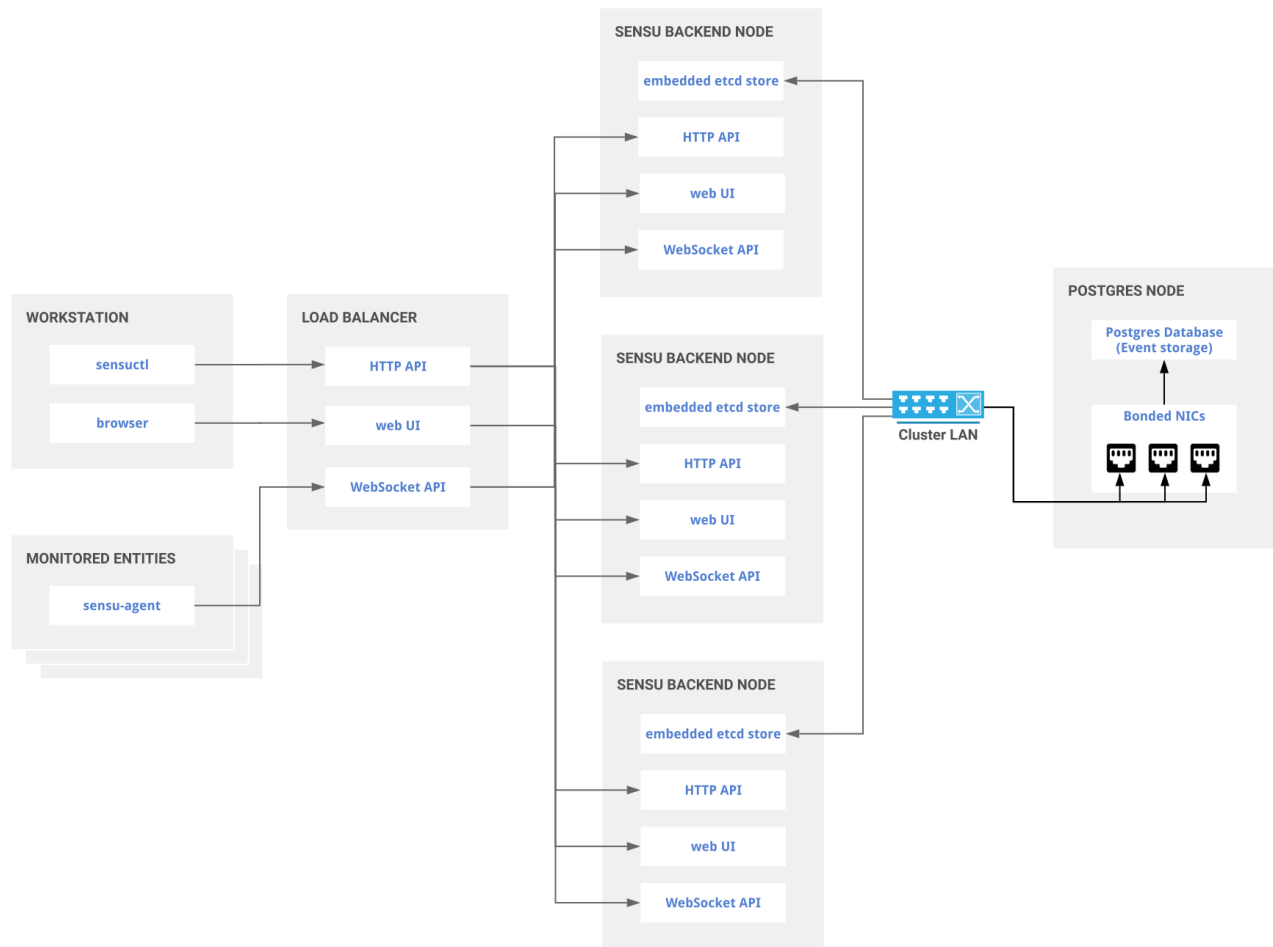
Clustered Sensu Go architecture with PostgreSQL event storage

In load testing, Sensu Go has proven capable of processing 36,000 events per second when using PostgreSQL as the event store.See the sensu-perf project repository for a detailed explanation of our testing methodology and results.

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 shorder 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.

Run a Sensu cluster

A Sensu cluster is a group of at least three sensu-backend nodes, each connected to a shared etcd cluster, using 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 will help you cope with the loss of a backend node, prevent data loss, and distribute the network load of agents.

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 flags are prefixed with `etcd` .For more detailed descriptions of the different arguments, see the etcd docs or the 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 (see 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 see examples of etcd command line flags. The configuration file entries in the rest of this guide translate to `sensu-backend` flags.

Sensu backend configuration

The examples in this section are configuration snippets from `/etc/sensu/backend.yml` using a three-node cluster. The nodes are named `backend-1`, `backend-2` and `backend-3` 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.

backend-1

```
##
# store configuration for backend-1/10.0.0.1
##
etcd-advertise-client-urls: "http://10.0.0.1:2379"
etcd-listen-client-urls: "http://10.0.0.1:2379"
etcd-listen-peer-urls: "http://0.0.0.0:2380"
etcd-initial-cluster: "backend-1=http://10.0.0.1:2380,backend-
2=http://10.0.0.2:2380,backend-3=http://10.0.0.3:2380"
```

```
etcd-initial-advertise-peer-urls: "http://10.0.0.1:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: ""
etcd-name: "backend-1"
```

backend-2

```
##
# store configuration for backend-2/10.0.0.2
##
etcd-advertise-client-urls: "http://10.0.0.2:2379"
etcd-listen-client-urls: "http://10.0.0.2:2379"
etcd-listen-peer-urls: "http://0.0.0.0:2380"
etcd-initial-cluster: "backend-1=http://10.0.0.1:2380,backend-2=http://10.0.0.2:2380,backend-3=http://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "http://10.0.0.2:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: ""
etcd-name: "backend-2"
```

backend-3

```
##
# store configuration for backend-3/10.0.0.3
##
etcd-advertise-client-urls: "http://10.0.0.3:2379"
etcd-listen-client-urls: "http://10.0.0.3:2379"
etcd-listen-peer-urls: "http://0.0.0.0:2380"
etcd-initial-cluster: "backend-1=http://10.0.0.1:2380,backend-2=http://10.0.0.2:2380,backend-3=http://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "http://10.0.0.3:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: ""
etcd-name: "backend-3"
```

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.

```
##
# backend-url configuration for all agents connecting to cluster over ws
##

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

        ID            Name                  Error                          Healthy
 ─────────────────── ──────────── ──────────────────────────────────────────────────────────────
 ──────────
 a32e8f613b529ad4   backend-1                                                true
```

```
c3d9f4b8d0dd1ac9  backend-2  dial tcp 10.0.0.2:2379: connect: connection refused  false
c8f63ae435a5e6bf  backend-3                                               true
```

Add a cluster member

Add a new member node to an existing cluster:

```
sensuctl cluster member-add backend-4 https://10.0.0.4:2380

added member 2f7ae42c315f8c2d to cluster

ETCD_NAME="backend-4"
ETCD_INITIAL_CLUSTER="backend-4=https://10.0.0.4:2380,backend-
1=https://10.0.0.1:2380,backend-2=https://10.0.0.2:2380,backend-
3=https://10.0.0.3:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

List cluster members

List the ID, name, peer URLs, and client URLs of all nodes in a cluster:

```
sensuctl cluster member-list

    ID          Name          Peer URLs          Client URLs
_____ _____ _____ _____

a32e8f613b529ad4  backend-1   https://10.0.0.1:2380   https://10.0.0.1:2379
c3d9f4b8d0dd1ac9  backend-2   https://10.0.0.2:2380   https://10.0.0.2:2379
c8f63ae435a5e6bf  backend-3   https://10.0.0.3:2380   https://10.0.0.3:2379
2f7ae42c315f8c2d  backend-4   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

Removed member 2f7ae42c315f8c2d from cluster
```

Replace a faulty cluster member

To replace a faulty cluster member to restore a cluster's health, start by running `sensuctl cluster health` to identify the faulty cluster member. For a faulty cluster member, the `Error` column will include an error message and the `Healthy` column will list `false`.

In this example, cluster member `backend-4` is faulty:

```
sensuctl cluster health

  ID               Name          Error                              Healthy
_____ _____ _____
_____

a32e8f613b529ad4   backend-1                                          true
c3d9f4b8d0dd1ac9   backend-2                                          true
c8f63ae435a5e6bf   backend-3                                          true
2f7ae42c315f8c2d   backend-4   dial tcp 10.0.0.4:2379: connect: connection refused   false
```

Then, delete the faulty cluster member. To continue this example, you will delete cluster member `backend-4` using its ID:

```
sensuctl cluster member-remove 2f7ae42c315f8c2d

Removed member 2f7ae42c315f8c2d from cluster
```

Finally, add a newly created member to the cluster. You can use the same name and IP address as the faulty member you deleted, with one change to the configuration: specify the `etcd-initial-cluster-state` as `existing`.

```
etcd-advertise-client-urls: "http://10.0.0.4:2379"
etcd-listen-client-urls: "http://10.0.0.4:2379"
```

```
etcd-listen-peer-urls: "http://0.0.0.0:2380"
etcd-initial-cluster: "backend-1=http://10.0.0.1:2380,backend-
2=http://10.0.0.2:2380,backend-3=http://10.0.0.3:2380,backend-
4=http://10.0.0.4:2380"
etcd-initial-advertise-peer-urls: "http://10.0.0.4:2380"
etcd-initial-cluster-state: "existing"
etcd-initial-cluster-token: ""
etcd-name: "backend-4"
```

If replacing the faulty cluster member does not resolve the problem, see 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

Updated member with ID c8f63ae435a5e6bf in cluster
```

Cluster security

See Secure Sensu for information about cluster security.

Use an external etcd cluster

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.

In this example, you will enable client-to-server and peer communication authentication using self-signed TLS certificates. To start etcd for `backend-1` 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=https://10.0.0.1:2380,backend-
2=https://10.0.0.2:2380,backend-3=https://10.0.0.3:2380" \
--initial-advertise-peer-urls "https://10.0.0.1:2380" \
--initial-cluster-state "new" \
--name "backend-1" \
--trusted-ca-file=./ca.pem \
--cert-file=./backend-1.pem \
--key-file=./backend-1-key.pem \
--client-cert-auth \
--peer-trusted-ca-file=./ca.pem \
--peer-cert-file=./backend-1.pem \
--peer-key-file=./backend-1-key.pem \
--peer-client-cert-auth \
--auto-compaction-mode revision \
--auto-compaction-retention 2
```

NOTE: *The* `auto-compaction-mode` *and* `auto-compaction-retention` *flags are important. Without these settings, your database may quickly reach etcd's maximum database size limit.*

To tell Sensu to use this external etcd data source, add the `sensu-backend` flag `--no-embed-etcd` to the original configuration, along with the path to a client certificate created using your CA:

```
sensu-backend start \
--etcd-trusted-ca-file=./ca.pem \
--etcd-cert-file=./client.pem \
--etcd-key-file=./client-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
```

Troubleshoot clusters

Failure modes

See the etcd failure modes documentation for information about cluster failure modes.

Disaster recovery

See the etcd recovery guide for disaster recovery information.

Scale Sensu Go with Enterprise datastore

COMMERCIAL FEATURE: Access the datastore feature in the packaged Sensu Go distribution. For more information, see [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.

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

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:

```
$ sudo -u postgres psql
postgres=# CREATE DATABASE sensu_events;
CREATE DATABASE
postgres=# CREATE USER sensu WITH ENCRYPTED PASSWORD 'mypass';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE sensu_events TO sensu;
GRANT
postgres-# \q
```

With this configuration complete, Postgres 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:

```
# make a copy of the current pg_hba.conf
sudo cp /var/lib/pgsql/data/pg_hba.conf /var/tmp/pg_hba.conf.bak
# give sensu user permissions to connect to 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
# restart postgresql service to activate pg_hba.conf changes
sudo systemctl restart postgresql
```

With this configuration complete, you can configure Sensu to store events in your Postgres database.

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
  }
```

```
    }
```

This configuration is written to disk as `my-postgres.yml` , and you can install it using `sensuctl` :

```
sensuctl create -f my-postgres.yml
```

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.This code illustrates connecting to the `sensu_events` database, listing the tables in the database, and requesting a list of all entities reporting keepalives:

```
postgres=# \c sensu_events
You are now connected to database "sensu_events" as user "postgres".
sensu_events=# \dt
            List of relations
 Schema |        Name        | Type  | Owner
--------+--------------------+-------+-------
 public | events             | table | sensu
 public | migration_version  | table | sensu
```

```
(2 rows)
 sensu_events=# select sensu_entity from events where sensu_check = 'keepalive';
  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 contains the same configuration you used to configure the Enterprise event store
earlier in this guide:

```
sensuctl delete -f my-postgres.yml
```

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 Postgres, switching back to the etcd datastore does not migrate current event data
from one store to another. You may see old events in the web UI or sensuctl output until the etcd
datastore catches up with the current state of your monitored infrastructure.

Configure Postgres streaming replication

Postgres supports an active standby by using streaming replication. All Sensu events written to the
primary Postgres server will be replicated to the standby server.

NOTE: Paths and service names may vary based on your operating system.

This section describes how to configure PostgreSQL streaming replication in four steps.

Step 1: Create and add the replication role

If you have administrative access to Postgres, you can create the replication role:

```
$ sudo -u postgres psql
postgres=# CREATE ROLE repl PASSWORD 'secret' LOGIN REPLICATION;
CREATE ROLE
postgres-# \q
```

Then, you must 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
```

Next, give the repl user permissions to replicate from the standby host. In the following command, replace `STANDBY_IP` with the IP address of your standby host:

```
export STANDBY_IP=192.168.52.10
echo "host replication repl ${STANDYB_IP}/32 md5" | sudo tee -a
/var/lib/pgsql/data/pg_hba.conf
```

Restart the PostgreSQL service to activate the `pg_hba.conf` changes:

```
sudo systemctl restart postgresql
```

Step 2: Set streaming replication configuration parameters

The next step is to set the streaming replication configuration parameters on the primary Postgres host.

Begin by making a copy of the `postgresql.conf` :

```
sudo cp -a /var/lib/pgsql/data/postgresql.conf
/var/lib/pgsql/data/postgresql.conf.bak
```

Next, append the necessary configuration options.

```
echo 'wal_level = hot_standby' | sudo tee -a /var/lib/pgsql/data/postgresql.conf
```

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
```

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 may not be necessary._

Restart the PostgreSQL service to activate the `postgresql.conf` changes:

```
sudo systemctl restart postgresql
```

Step 3: Bootstrap the standby host

The standby host must be bootstrapped using the `pg_basebackup` command. This process will copy all configuration files from the primary as well as databases.

If the standby host has ever run Postgres, you must empty the data directory:

```
sudo systemctl stop postgresql
sudo mv /var/lib/pgsql/data /var/lib/pgsql/data.bak
```

Make the standby data directory:

```
sudo install -d -o postgres -g postgres -m 0700 /var/lib/pgsql/data
```

And then bootstrap the standby data directory:

```
export PRIMARY_IP=192.168.52.11
sudo -u postgres pg_basebackup -h $PRIMARY_IP -D /var/lib/pgsql/data -P -U repl -R -
-xlog-method=stream
Password:
30318/30318 kB (100%), 1/1 tablespace
```

Step 4: Confirm replication

To confirm your configuration is working properly, start by removing configurations that are only for the primary:

```
sudo sed -r -i.bak '/^(wal_level|max_wal_senders|wal_keep_segments).*/d'
/var/lib/pgsql/data/postgresql.conf
```

Start the PostgreSQL service:

```
sudo systemcl start postgresql
```

To verify that the replication is taking place, check the commit log location on the primary and standby

hosts:

```
# From master
sudo -u postgres psql -c "select pg_current_xlog_location()"
 pg_current_xlog_location
-------------------------
 0/3000568
(1 row)
# From standby
sudo -u postgres psql -c "select pg_last_xlog_receive_location()"
 pg_last_xlog_receive_location
-------------------------------
 0/3000568
(1 row)

# From standby
sudo -u postgres psql -c "select pg_last_xlog_replay_location()"
 pg_last_xlog_replay_location
------------------------------
 0/3000568
(1 row)
```

With this configuration complete, your Sensu events will be replicated to the standby host.

Secure Sensu

As with any piece of software, it is critical to minimize any attack surface the software exposes. Sensu is no different. This guide describes the components you need to secure to make Sensu production-ready.

Secure etcd peer communication

You can secure etcd peer communication via the configuration at `/etc/sensu/backend.yml`. Here are the parameters you'll need to configure:

```
##
# backend store configuration
##
etcd-listen-client-urls: "https://localhost:2379"
etcd-listen-peer-urls: "https://localhost:2380"
etcd-initial-advertise-peer-urls: "https://localhost:2380"
etcd-cert-file: "/path/to/your/cert"
etcd-key-file: "/path/to/your/key"
etcd-trusted-ca-file: "/path/to/your/ca/file"
etcd-peer-cert-file: "/path/to/your/peer/cert"
etcd-peer-key-file: "/path/to/your/peer/key"
etcd-peer-client-cert-auth: "true"
etcd-peer-trusted-ca-file: "/path/to/your/peer/ca/file"
```

Secure the API and dashboard

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. Here are the attributes you'll need to configure.

NOTE: *By changing these parameters, the server will communicate over TLS and expect agents that connect to it to use the WebSocket secure protocol. For communication to continue, you must complete the steps in this section and in the Secure Sensu agent-to-server communication section.*

```
##
# backend ssl configuration
##
cert-file: "/path/to/ssl/cert.pem"
key-file: "/path/to/ssl/key.pem"
trusted-ca-file: "/path/to/trusted-certificate-authorities.pem"
insecure-skip-tls-verify: false
```

Providing these cert-file and key-file parameters will cause the Agent Websocket API and HTTP API to serve requests over SSL/TLS (https). As a result, you will also need to specify `https://` schema for the `api-url` parameter:

```
##
# backend api configuration
##
api-url: "https://localhost:8080"
```

You can also specify a certificate and key for the dashboard separately from the API using the `dashboard-cert-file` and `dashboard-key-file` parameters:

```
##
# backend ssl configuration
##
cert-file: "/path/to/ssl/cert.pem"
key-file: "/path/to/ssl/key.pem"
trusted-ca-file: "/path/to/trusted-certificate-authorities.pem"
insecure-skip-tls-verify: false
```

```
dashboard-cert-file: "/path/to/ssl/cert.pem"
dashboard-key-file: "/path/to/ssl/key.pem"
```

In this example, we provide the path to the cert, key, and CA file. After you restart the `sensu-backend` service, the parameters will load and you will able to access the dashboard at https://localhost:3000. Configuring these attributes will also ensure that agents can communicate securely.

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** in the Secure the API and dashboard section.*

By default, an agent uses the insecure `ws://` transport. Here's an example from `/etc/sensu/agent.yml`:

```
---
##
# agent configuration
##
backend-url:
  - "ws://127.0.0.1:8081"
```

To use WebSocket over SSL/TLS (wss), change the `backend-url` value to the `wss://` schema:

```
---
##
# agent configuration
##
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 as part of the agent configuration by passing `--trusted-ca-file` if

you are starting the agent via `sensu-agent start`. You may include it as part of the agent
configuration in `/etc/sensu/agent.yml`:

```
trusted-ca-file: "/path/to/trusted-certificate-authorities.pem"
```

*NOTE: If you are creating a Sensu cluster, every cluster member needs to be present in the
configuration. See Run a Sensu cluster for more information about how to configure agents for a
clustered configuration.*

Sensu agent TLS authentication

COMMERCIAL FEATURE: Access client TLS authentication in the packaged Sensu Go distribution.
For more information, see 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, see the RBAC reference and Create a read-only user.

Sensu can also use TLS authentication for connecting agents to backends. When agent TLS
authentication is enabled, agents do not need to send password credentials to backends when they
connect. In addition, when using TLS authentication, agents do not require an explicit user in Sensu.
They will default to using the `system:agents` group.

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.*

To view a certificate's CN with openssl:

```
$ openssl x509 -in client.pem -text -noout
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 TLS authentication, use existing certificates and keys for the Sensu backend and agent or create new certificates and keys according to the <u>Create self-signed certificates</u> section.

After you create backend and agent certificates, modfiy the backend and agent configuration:

```
##
# backend configuration
##
agent-auth-cert-file: "/path/to/backend-1.pem"
agent-auth-key-file: "/path/to/backend-1-key.pem"
agent-auth-trusted-ca-file: "/path/to/ca.pem"
```

```
##
# agent configuration
##
cert-file: "/path/to/agent-1.pem"
key-file: "/path/to/agent-1-key.pem"
trusted-ca-file: "/path/to/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.

Create self-signed certificates for securing etcd and backend-agent communication

This example uses the cfssl tool to generate self-signed certificates.

First, create a Certificate Authority (CA). To keep things straightforward in this example, you will generate all clients and peer certificates using this CA, but you might eventually want to create a distinct CA.

```
echo '{"CN":"CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert -initca - |
cfssljson -bare ca -
echo '{"signing":{"default":{"expiry":"43800h","usages":["signing","key
encipherment","server auth","client auth"]}}}' > ca-config.json
```

Then, using that CA, generate certificates and keys for each peer (backend server) by specifying their Common Name (CN) and their hosts. A `*.pem`, `*.csr` and `*.pem` will be created for each backend.

```
export ADDRESS=10.0.0.1,backend-1
export NAME=backend-1
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="$ADDRESS" -
profile=peer - | cfssljson -bare $NAME

export ADDRESS=10.0.0.2,backend-2
export NAME=backend-2
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="$ADDRESS" -
profile=peer - | cfssljson -bare $NAME

export ADDRESS=10.0.0.3,backend-3
export NAME=backend-3
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="$ADDRESS" -
profile=peer - | cfssljson -bare $NAME
```

You will also create generate a *client* certificate that can be used by clients to connect to the etcd client URL. This time, you don't need to specify an address, only a CN (here, `client`). The files `client-key.pem`, `client.csr`, and `client.pem` will be created:

```
export NAME=client
```

```
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=client - | cfssljson -bare $NAME
```

If you have a Sensu license, you can also generate a certificate for agent TLS authentication:

```
export NAME=agent-1
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=client - | cfssljson -bare $NAME
```

See etcd's guide to generating self-signed certificates for detailed instructions for securing etcd.

When you're finished, the following files should be created (the `*.csr` files are not used in this guide):

```
agent-1-key.pem
agent-1.csr
agent-1.pem
backend-1-key.pem
backend-1.csr
backend-1.pem
backend-2-key.pem
backend-2.csr
backend-2.pem
backend-3-key.pem
backend-3.csr
backend-3.pem
ca-config.json
ca-key.pem
ca.csr
ca.pem
client-key.pem
client.csr
client.pem
```

Next steps

Learn about role-based access control (RBAC) in Sensu or create a read-only user.

Use the API key feature

-

-

The Sensu API key feature (core/v2.APIKey) is a persistent UUID that maps to a stored Sensu username. The advantages of authenticating with API keys rather than <u>access tokens</u> 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
```

Example

```
$ curl -H "Authorization: Key 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks

HTTP/1.1 200 OK
[
  {
    "command": "check-cpu.sh -w 75 -c 90",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "publish": true,
    "subscriptions": [
      "linux"
    ],
    "metadata": {
      "name": "check-cpu",
      "namespace": "default"
    }
  }
]
```

Sensuctl management commands

NOTE: *The API key resource is intentionally not compatible with* `sensuctl create` *.*

To generate a new API key for the admin user:

```
$ sensuctl api-key grant admin
Created: /api/core/v2/apikeys/7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2
```

To get information about an API key:

```
$ sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format json
{
  "metadata": {
    "name": "7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2"
  },
  "username": "admin",
  "created_at": 1570744117
}
```

To get a list of all API keys:

```
$ sensuctl api-key list
            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
Deleted
```

Create a read-only user with RBAC

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
   ```

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
   ```

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
```

You can also use role bindings to tie roles directly to users using the `--user` flag.

All users in the `ops` group now have read-only access to all resources within the default namespace. You can use the `sensuctl user`, `sensuctl role`, and `sensuctl role-binding` commands to manage your RBAC configuration.

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
```

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
```

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
```

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.

Plan maintenance windows with silencing

As the Sensu backend processes check results, the server executes [event handlers](#) to send alerts to personnel or otherwise relay event data to external services. Sensu's built-in silencing, along with the built-in `not_silenced` filter, provides a way to suppress execution of event handlers on an ad hoc basis.

Use silencing to prevent handlers configured with the `not_silenced` filter from being triggered based on the check name in a check result or the subscriptions associated with the entity that published the check result. Sensu's silencing capability allows operators to quiet incoming alerts while coordinating a response or during planned maintenance windows.

Sensu silencing makes it possible to:

￢ [Silence all checks on a specific entity](#)

￢ [Silence a specific check on a specific entity](#)

￢ [Silence all checks on entities with a specific subscription](#)

￢ [Silence a specific check on entities with a specific subscription](#)

￢ [Silence a specific check on every entity](#)

Use silencing to plan maintenance

Suppose you want to plan a maintenance window. In this example, you'll create a silenced entry for a specific entity named `i-424242` and its check, `check-http`, to prevent alerts as you restart and redeploy the services associated with this entity.

Create the silenced entry

To begin, create a silenced entry that will silence the check `check-http` on the entity `i-424242` for a planned maintenance window that starts at **01:00** on **Sunday** and ends **1 hour** later. Your username will be added automatically as the **creator** of the silenced entry:

```
sensuctl silenced create \
--subscription 'entity:i-424242' \
--check 'check-http' \
--begin '2018-03-16 01:00:00 -04:00' \
--expire 3600 \
--reason 'Server upgrade'
```

See the sensuctl documentation for the supported time formats for the `begin` flag.

Validate the silenced entry

Use sensuctl to verify that the silenced entry against the entity `i-424242` was created properly:

```
sensuctl silenced info 'entity:i-424242:check-http'
```

After the silenced entry starts to take effect, events that are silenced will be marked as such in `sensuctl events` :

```
sensuctl event list

  Entity        Check       Output     Status   Silenced       Timestamp
─────────────── ────────── ────────── ──────────── ──────────
──────────────────────────────
  i-424242    check-http              0       true    2018-03-16 13:22:16 -0400 EDT
```

WARNING: *By default, a silenced event will be handled unless the handler uses the* `not_silenced` *filter to discard silenced events.*

Next steps

Next, read the silencing reference for in-depth documentation about silenced entries.

Troubleshoot

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 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

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 if you change log levels via configuration files or command line arguments. For help with restarting a service, see the agent reference or backend reference.

> **NOTE**: You can set the log level to `trace`, but Sensu does not log at the trace level. The trace log level is the same as `debug` in Sensu.

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/`, see forwarding logs from journald to syslog.

The following table lists the common targets for logging and example commands for following those logs. You may substitute the name of the desired service (e.g. `backend` or `agent`) for the `${service}` variable.

Platform	Version	Target	Command to follow log
RHEL/Centos	>= 7	journald	`journalctl --follow --unit sensu-${service}`
RHEL/Centos	<= 6	log file	

| | | | ```
tail --follow
/var/log/sensu/sensu-${service}
``` |
| Ubuntu | >= 15.04 | journald | ```
journalctl --follow --unit
sensu-${service}
``` |
| Ubuntu | <= 14.10 | log file | ```
tail --follow
/var/log/sensu/sensu-${service}
``` |
| Debian | >= 8 | journald | ```
journalctl --follow --unit
sensu-${service}
``` |
| Debian | <= 7 | log file | ```
tail --follow
/var/log/sensu/sensu-${service}
``` |

NOTE: *Platform versions are listed for reference only and do not supersede the documented* <u>*supported platforms*</u>.

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 since yesterday:

```
journalctl _COMM=sensu-backend.service --since yesterday | tee sensu-backend-$(date
+%Y-%m-%d).log
```

Retrieve all the logs for Sensu since a specific time:

```
journalctl _COMM=sensu-backend.service --since 09:00 --until "1 hour ago" | tee
sensu-backend-$(date +%Y-%m-%d).log
```

Retrieve all the logs for Sensu for a specific date range:

```
journalctl _COMM=sensu-backend.service --since "2015-01-10" --until "2015-01-11
03:00" | tee sensu-backend-$(date +%Y-%m-%d).log
```

Windows

The Sensu agent stores service logs to the location specified by the `log-file` configuration flag
(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, see 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
```

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
3.3","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, see etcd security documentation.

Permission issues

The Sensu user and group must own files and folders within `/var/cache/sensu/` and `/var/lib/sensu/`. You will see 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. 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
```

It may also be helpful to see 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 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.

Assets

Asset filters allow you to scope an asset to a particular operating system or architecture. You can see an example in 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

```json
{
    "asset": "check-disk-space",
    "component": "asset-manager",
    "entity": "sensu-centos",
    "filters": [
        "true == false"
    ],
    "level": "debug",
    "msg": "entity not filtered, not installing asset",
    "time": "2019-09-12T18:28:05Z"
}
```

Backend event

```json
{
 "timestamp": 1568148292,
 "check": {
    "command": "check-disk-space",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "sensu-plugins-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,
    "env_vars": null,
    "metadata": {
      "name": "failing-disk-check",
      "namespace": "default"
    }
  },
  "metadata": {
    "namespace": "default"
  }
}
```

If you see 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. You can review the filters for an asset by using the sensuctl `asset info` command with a `--format` flag:

```
sensuctl asset info sensu-plugins-disk-checks --format yaml
```

or

```
sensuctl asset info sensu-plugins-disk-checks --format json
```

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:

```
...
    - 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:

```
...
    - entity.system.platform_family == 'debian'
    - entity.system.arch == 'amd64'
```

or

```
    - entity.system.platform == 'ubuntu'
    - entity.system.arch == 'amd64'
```

This would allow the asset to be downloaded onto the target entity.

Multi-cluster visibility with federation

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

Sensu's federation API allows you to register external clusters, access resources across multiple clusters via the web UI, and mirror your changes in one cluster to follower clusters.

Federation is not enabled by default. You must create a cluster resource for the federation cluster and register it.

Create, update, and delete clusters using sensuctl create, edit, and delete commands. Only cluster administrators can register a new cluster, but every user can query the list of clusters.

What you can do with federation

Federation affords visibility into the health of your infrastructure and services across multiple distinct Sensu instances within a single web UI. 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.

Configure federation

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

The following steps are required to configure these features. Our scenario assumes that we 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 scenario, 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 comprised of multiple sensu-backends are supported.

Upon completion of these steps, you'll be able to browse events, entities, checks and other resources in the `gateway`, `alpha` and `beta` clusters from the `gateway` cluster web UI.

This diagram depicts the federation relationship documented in this guide:

Example federated architecture
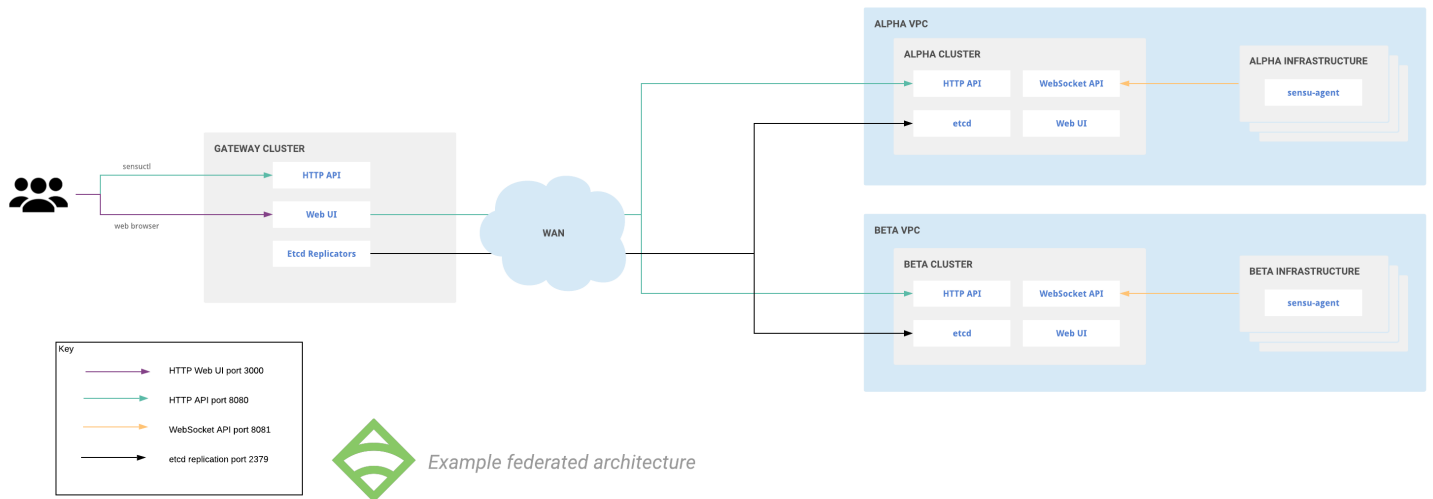
Step 1 Configure backends for TLS

Because federation depends on communication with multiple disparate clusters, working TLS is required for successful federated operation.

To ensure that cluster members can validate one another, certificates for each cluster member should include the IP addresses and/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.

To continue with this guide, make sure you have the required TLS credentials in place:

- ┐ PEM-formatted X.509 certificate and corresponding private key copied to each cluster member
- ┐ Corresponding CA certificate chain copied to each cluster member

If you don't have existing infrastructure for issuing certificates, see Secure Sensu for our recommended self-signed certificate issuance process.

This prerequisite extends to configuring the following Sensu backend etcd parameters:

Backend property	Note
`etcd-cert-file`	Path to certificate used for TLS on etcd client/peer communications.
`etcd-key-file`	Path to key corresponding with `etcd-cert-file` certificate.
`etcd-trusted-ca-`	Path to CA certificate chain file. This CA certificate chain must be usable

`file`	to validate certificates for all backends in the federation.
`etcd-client-cert-auth`	Enforces certificate validation to authenticate etcd replicator connections. We recommend setting to `true`.
`etcd-advertise-client-urls`	List of https URLs to advertise for etcd replicators, accessible by other backends in the federation (e.g. `https://sensu.beta.example.com:2379`).
`etcd-listen-client-urls`	List of https URLs to listen on for etcd replicators (e.g. `https://0.0.0.0:2379` to listen on port 2379 across all ipv4 interfaces).

NOTE: You must provide non-default values for the `etcd-advertise-client-urls` and `etcd-listen-client-urls` backend parameters. The default values are not suitable for use under federation.

Step 2 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. This use of 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 need to generate keys as files on disk so they can be copied to all backends in the federation.

Use 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
```

Then generate a public key from the private key:

```
openssl ec -in jwt_private.pem -pubout -out jwt_public.pem
```

For this example, you'll put JWT keys into `/etc/sensu/certs` on each cluster backend, and use the `jwt-private-key-file` and `jwt-public-key-file` attributes in `/etc/sensu/backend.yml` to specify the paths to these JWT keys:

```
jwt-private-key-file: /etc/sensu/certs/jwt_private.pem
jwt-public-key-file: /etc/sensu/certs/jwt_public.pem
```

After updating the backend configuration in each cluster, restart `sensu-backend` so that your settings take effect:

```
sensu-backend start
```

Step 3 Add a cluster role binding and user

To test your configuration, provision a User and a ClusterRoleBinding in the `gateway` cluster.

First, confirm that sensuctl is configured to communicate with the `gateway` cluster using `sensuctl config view` to see the active configuration:

```
=== Active Configuration
API URL:    https://sensu.gateway.example.com:8080
Namespace: default
Format:    tabular
Username:  admin
```

Second, create a `federation-viewer` user:

```
sensuctl user create federation-viewer --interactive
```

When prompted, enter a password for the `federation-viewer` user.When prompted for groups,

press enter. Note the `federation-viewer` 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.

Next, grant the `federation-viewer` user read-only access through 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
```

In step 4, you'll configure etcd replicators to copy the cluster role bindings and other RBAC policies you created in the `gateway` cluster to the `alpha` and `beta` clusters.

Step 4 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 those resources to other clusters in the federation, ensuring consistent permissions for Sensu users across multiple clusters via the `gateway` web UI.

To get started, configure one etcd replicator per cluster for each of those RBAC policy types, 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 resources that belong to those namespaces.

For example, these etcd replicator resources will replicate ClusterRoleBinding resources from the `gateway` cluster to 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/cert.pem"
key: "/etc/sensu/certs/key.pem"
url: https://sensu.alpha.example.com:2379
api_version: core/v2
resource: ClusterRoleBinding
replication_interval_seconds: 30
```

JSON

```json
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "AlphaClusterRoleBindings"
  },
  "spec": {
    "ca_cert": "/etc/sensu/certs/ca.pem",
    "cert": "/etc/sensu/certs/cert.pem",
    "key": "/etc/sensu/certs/key.pem",
    "url": "https://sensu.alpha.example.com:2379",
    "api_version": "core/v2",
    "resource": "ClusterRoleBinding",
    "replication_interval_seconds": 30
  }
}
```

YML

```yaml
---
api_version: federation/v1
type: EtcdReplicator
metadata:
  name: BetaClusterRoleBindings
spec:
  ca_cert: "/etc/sensu/certs/ca.pem"
  cert: "/etc/sensu/certs/cert.pem"
  key: "/etc/sensu/certs/key.pem"
  url: https://sensu.beta.example.com:2379
  api_version: core/v2
  resource: ClusterRoleBinding
```

```
      replication_interval_seconds: 30
```

JSON

```json
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "BetaClusterRoleBindings"
  },
  "spec": {
    "ca_cert": "/etc/sensu/certs/ca.pem",
    "cert": "/etc/sensu/certs/cert.pem",
    "key": "/etc/sensu/certs/key.pem",
    "url": "https://sensu.beta.example.com:2379",
    "api_version": "core/v2",
    "resource": "ClusterRoleBinding",
    "replication_interval_seconds": 30
  }
}
```

To configure this etcd replicator on your `gateway` cluster, use `sensuctl config view` to verify that `sensuctl` is configured to talk to a `gateway` cluster API. Reconfigure `sensuctl` if needed.

Write these EtcdReplicator definitions written to disk and use `sensuctl create -f` to apply them to the `gateway` cluster.

For a consistent experience, repeat the `ClusterRoleBinding` example in this guide for `Role`, `RoleBinding` and `ClusterRole` resource types.The `etcd replicators reference` includes examples you can follow for `Role`, `RoleBinding`, `ClusterRole`, and `ClusterRoleBinding` resources.

To verify that the EtcdReplicator resource is working as expected, reconfigure `sensuctl` to communicate with the `alpha` and then `beta` clusters, issuing the `sensuctl cluster-role-binding list` command for each.You should see the `federation-viewer-readonly` binding created in step 3 listed in the output from each cluster:

```
$ sensuctl cluster-role-binding info federation-viewer-readonly
=== federation-viewer-readonly
Name:          federation-viewer-readonly
```

```
Cluster Role: view
Subjects:
  Users:        federation-viewer
```

Step 5 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 which are members of the same etcd cluster. Creating a Cluster resource which lists multiple backends not belonging 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

```yaml
api_version: federation/v1
type: Cluster
metadata:
  name: alpha
spec:
  api_urls:
  - https://sensu.alpha.example.com:8080
```

JSON

```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` .

Step 6 Get a unified view of all your clusters in the web UI

After you create clusters using the federation API, 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:
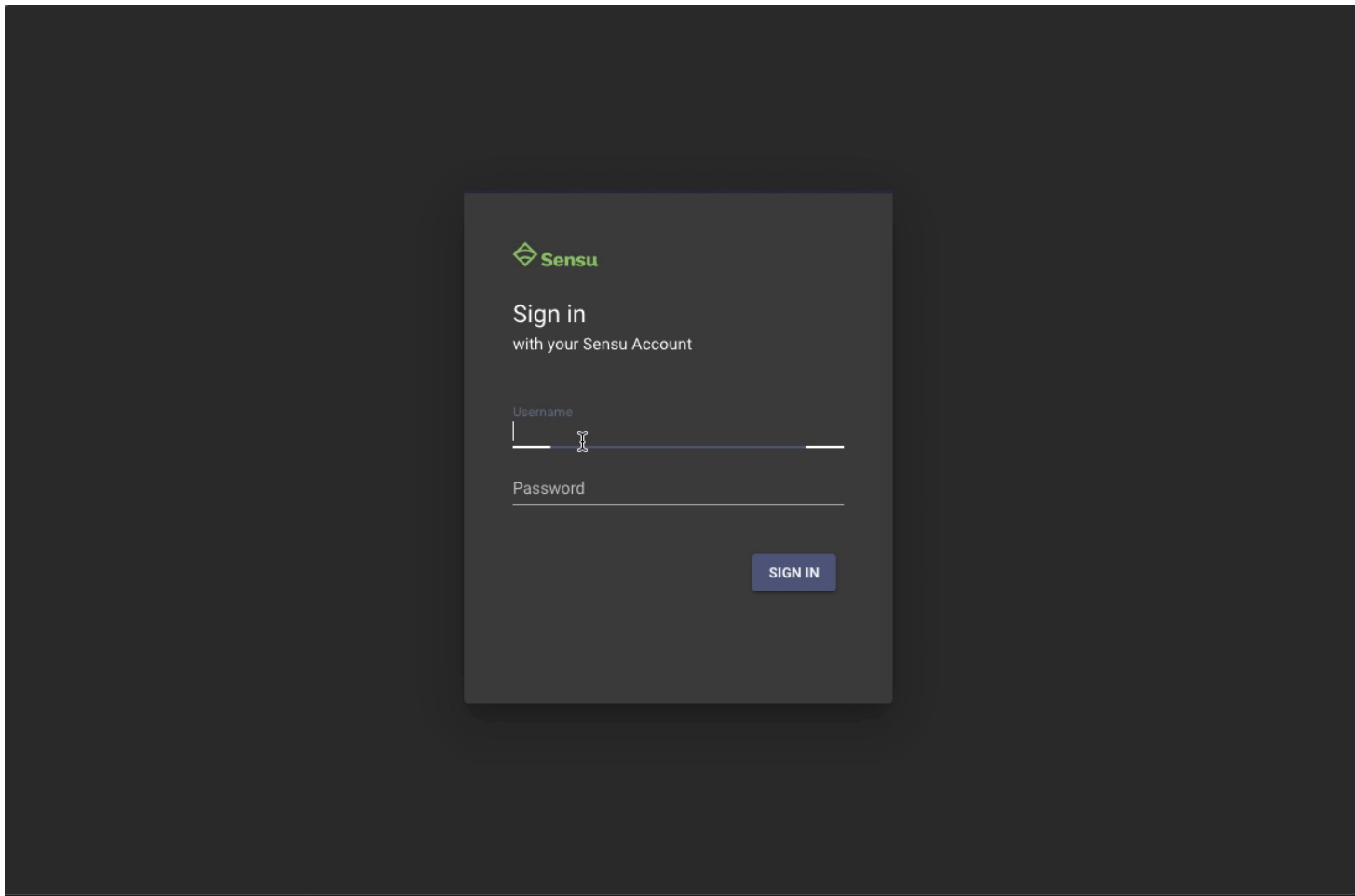
Because the `federation-viewer` user is granted only permissions provided by the built-in `view` role, this user should be able to view all resources across all clusters but should not be able to make any changes. If you haven't changed the permissions of the default `admin` user, that user should be able to view, create, delete, and update resources across all clusters.

Next steps

Learn more about configuring RBAC policies in our RBAC reference documentation.

Dashboard overview

- ¬ [Access the dashboard](#)

- ¬ [Sign in](#)

- ¬ [Themes](#)

- ¬ [Namespaces](#)

    - ¬ [Namespace switcher](#)

- ¬ [Manage entities](#)

- ¬ [Manage checks](#)

- ¬ [Manage handlers](#)

- ¬ [Manage event filters](#)

- ¬ [Manage mutators](#)

The Sensu backend includes the **Sensu dashboard**: a unified view of your events, entities, and checks with user-friendly tools to reduce alert fatigue.

COMMERCIAL FEATURE: Access the Sensu dashboard homepage (shown below) in the packaged Sensu Go distribution. For more information, see [Get started with commercial features](#).
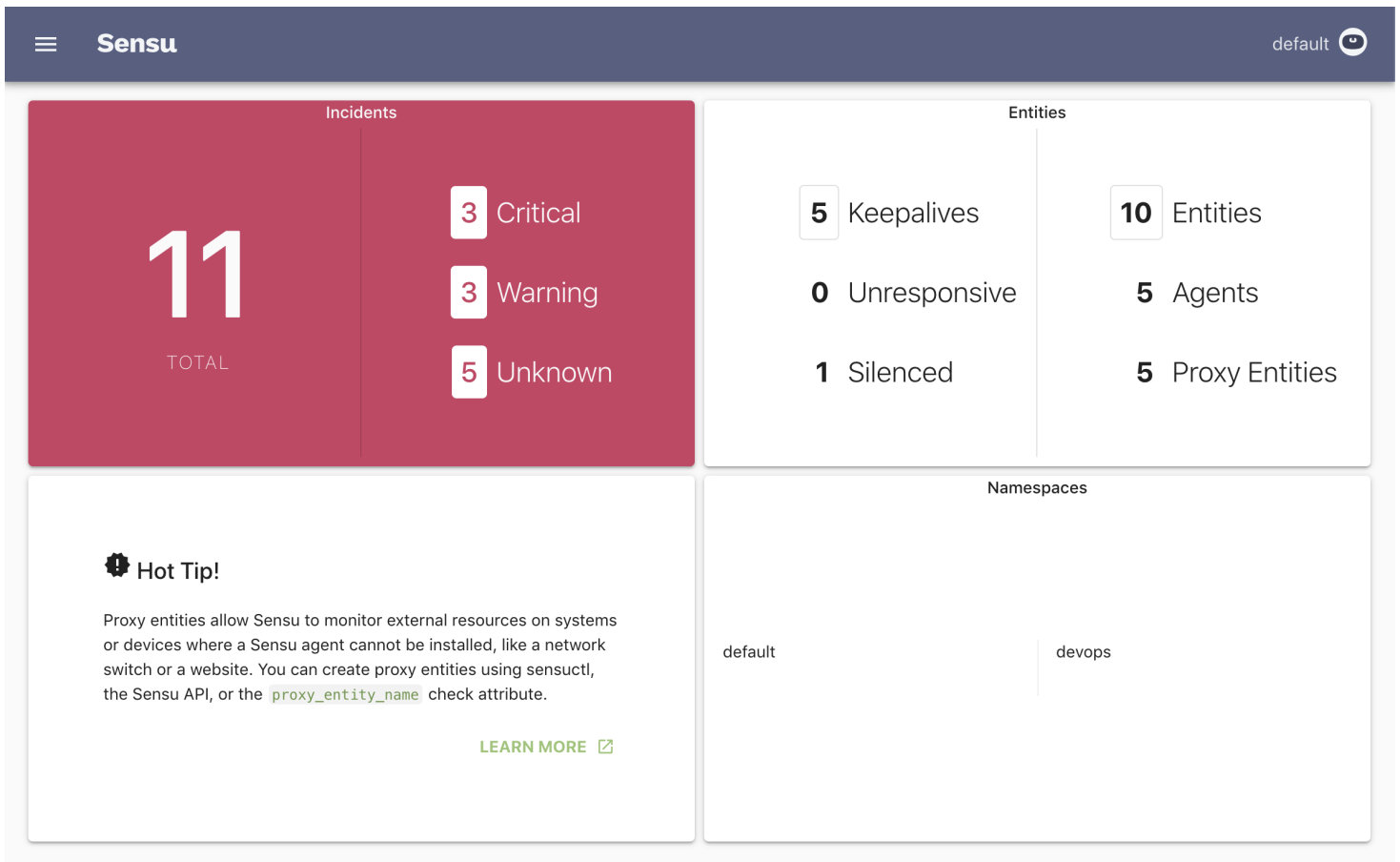
Sensu dashboard homepage

Access the dashboard

After you start the Sensu backend, you can access the dashboard 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

Sign in to the dashboard with your sensuctl username and password.See the role-based access contro
reference for default user credentials and instructions for creating new users.

Themes

Use the preferences menu to change the theme or switch to the dark theme.

Namespaces

The dashboard displays events, entities, checks, and silences for a single namespace at a time.By default, the dashboard displays the `default` namespace.

Namespace switcher

To switch namespaces, select the menu icon in the upper-left corner and choose a namespace from the dropdown.

COMMERCIAL FEATURE: In the packaged Sensu Go distribution, the namespace switcher will list only the namespaces to which the current user has access. For more information, see Get started with commercial features.
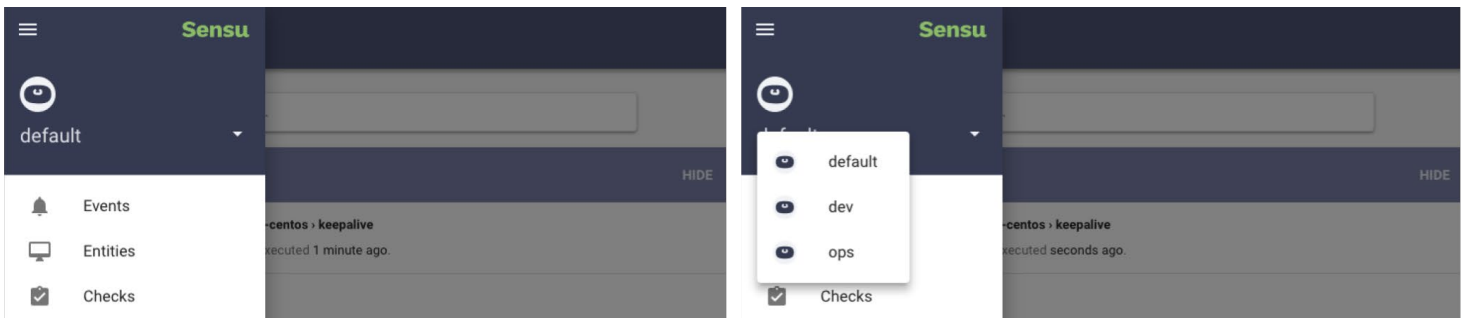
Sensu dashboard namespace switcher

Manage entities

You can delete Sensu entities in the dashboard Entities page.

Manage checks

COMMERCIAL FEATURE: Access check management in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

You can create, edit, and delete Sensu checks in the dashboard Checks page.

Manage handlers

COMMERCIAL FEATURE: Access handler management in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

You can create, edit, and delete Sensu handlers in the dashboard Handlers page.

Manage event filters

COMMERCIAL FEATURE: Access filter management in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

You can create, edit, and delete Sensu event filters in the dashboard Filters page.

Manage mutators

COMMERCIAL FEATURE: Access mutator management in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

You can create, edit, and delete Sensu mutators in the dashboard Mutators page.

Dashboard filtering

The Sensu dashboard includes basic filters you can use to build customized views of your Sensu resources. When applied, filters create a unique link, so you can bookmark and share your favorite filter combinations.

- On the **Events page**, filter by entity, check, status, and silenced/unsilenced. You can also sort events by severity, last OK, newest, and oldest.

- On the **Entities page**, filter by entity class and subscription.

- On the **Checks page**, filter by subscription and published/unpublished.

- On the **Handlers page**, filter by handler type.

- On the **Filters page**, filter by action.

- On the **Silences page**, filter by check and subscription. You can also sort silences by start date.

Advanced filtering

COMMERCIAL FEATURE: Access advanced filtering in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

Sensu supports advanced dashboard filtering using a wider range of attributes, including custom labels Select the filter bar to start building custom views using suggested attributes and values.

Filter with label selectors

To filter by custom labels on the Entities, Checks, Handlers, Filters, Mutators, and Silences pages, use the `labelSelector` filter and the `==` operator.

For example, on the **Entities page**, to display only entities with a `region: us-west-1` label:

```
labelSelector: region == "us-west-1"
```

Filter with field selectors

Field selectors let you fine-tune filters using the complete set of attributes supported by API filtering. For a complete list of supported attributes and operators, see the API docs.

For example, on the **Events page**, to display only events with the subscription `webserver` :

```
fieldSelector: webserver in event.check.subscriptions
```

On the **Checks page**, to display only checks that use the `slack` asset:

```
fieldSelector: slack in check.handlers
```

API overview

API version: v2

The Sensu backend REST API provides access to Sensu workflow configurations and monitoring event data. For information about the Sensu agent API, see the [agent reference](#).

URL format

Sensu API endpoints use the standard URL format
`/api/{group}/{version}/namespaces/{namespace}` where:

- ¬ `{group}` is the API group: `core`.
- ¬ `{version}` is the API version: `v2`.

- ⌐ `{namespace}` is the namespace name. The examples in these API docs 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. See the RBAC reference for more information about configuring Sensu users and access controls.

NOTE: The authentication API, authentication providers API, and health API do not follow this standard URL format.

Data format

The Sensu API uses JSON-formatted requests and responses. In terms of sensuctl output types, the Sensu API uses the `json` format, not `wrapped-json`.

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

API request bodies are limited to 0.512 MB in size.

Access control

With the exception of the health API and metrics API, the Sensu API requires authentication using a JSON Web Token (JWT) access token. Use the authentication API and your Sensu username and password to generate access tokens and refresh tokens. The Sensu API docs use `$SENSU_ACCESS_TOKEN` to represent a valid access token in API requests.

Authentication quickstart

To set up a local API testing environment, save your Sensu credentials and token as environment variables:

```
# Requires curl and jq
export SENSU_USER=YOUR_USERNAME && SENSU_PASS=YOUR_PASSWORD

export SENSU_ACCESS_TOKEN=`curl -X GET -u "$SENSU_USER:$SENSU_PASS" -s
http://localhost:8080/auth | jq -r ".access_token"`
```

Basic authentication using the authentication API

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..."
    }
    ```

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 see 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 token using sensuctl

You can also generate an API access token using the sensuctl command line tool. The user credentials that you use to log in to sensuctl determine your permissions to get, list, create, update, and delete resources with the Sensu API.

1. Install and log in to 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 see 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 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
```

Example

```
$ curl -H "Authorization: Key 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks

HTTP/1.1 200 OK
[
  {
    "command": "check-cpu.sh -w 75 -c 90",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "publish": true,
    "subscriptions": [
      "linux"
    ],
    "metadata": {
      "name": "check-cpu",
      "namespace": "default"
    }
  }
]
```

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: Bearer
$SENSU_ACCESS_TOKEN"
```

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=L2RlZmF1bU2Vuc3UtTWFjjQ
\
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
```

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
  }
]
```

Response filtering

COMMERCIAL FEATURE: Access API response filtering in the packaged Sensu Go distribution. For more information, see 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 use label and field selectors in the Sensu dashboard, see* *dashboard filtering*.

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.

The label selector does not work with values that contain special characters like hyphens and underscores.

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.check.handlers` `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.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`
Role	`role.name` `role.namespace`
RoleBinding	`rolebinding.name` `rolebinding.namespace` `rolebinding.role_ref.name` `rolebinding.role_ref.type`
Silenced	`silenced.name` `silenced.namespace` `silenced.check` `silenced.creator` `silenced.expire_on_resolve` `silenced.subscription`
User	`user.username` `user.disabled` `user.groups`

The `fieldSelector` parameter does not work with values that contain special characters like hyphens and underscores.

Operators

Sensu's API response filtering supports two equality-based operators, two set-based operators, 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`
`&&`	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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/checks -G \
--data-urlencode 'linux in check.subscriptions'
```

To retrieve checks that do not use the `slack` handler:

```
curl -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/checks -G \
--data-urlencode '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]`.

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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/checks -G \
--data-urlencode 'fieldSelector=linux in check.subscriptions' \
--data-urlencode 'labelSelector=type != "server"'
```

Examples

Use selectors with arrays of strings

To retrieve checks that are in either the `dev` or `production` namespace:

```
curl -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/events -G \
--data-urlencode 'fieldSelector=event.entity.status != "0"'
```

Filter checks, entities, or entities by subscription

To list all checks that include the `linux` subscription:

```
curl -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
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: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/events -G \
--data-urlencode 'fieldSelector=linux in event.entity.subscriptions'
```

APIKeys API

The `/apikeys` API endpoint

`/apikeys` (GET)

The `/apikeys` GET endpoint retrieves all API keys.

EXAMPLE

The following example demonstrates a request to the `/apikeys` API endpoint, resulting in a JSON array that contains all API keys.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/apikeys \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK

[
  {
    "metadata": {
      "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b"
    },
    "username": "admin",
```

```
      "created_at": 1570640363
    }
  ]
```

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. See 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"
},
"username": "admin",
"created_at": 1570640363
}
]```

`/apikeys` (POST)

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. The request includes the API key definition in the request body and returns a successful HTTP `201 Created` response.

```
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


HTTP/1.1 201 Created
```

API Specification

/apikeys (POST)	
description	Creates a new API key, a Sensu-generated 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)

The `/apikeys/:apikey` API endpoint

`/apikeys/:apikey` (GET)

The `/apikeys/:apikey` GET endpoint retrieves the specified API key.

EXAMPLE

In the following example, querying the `/apikeys/:apikey` API returns the requested `:apikey` definition or an error if the key is not found.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b"
  },
  "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"
  },
  "username": "admin",
  "created_at": 1570640363
}
```

`/apikeys/:apikey` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b

HTTP/1.1 204 No Content
```

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)

Assets API

The `/assets` API endpoint

`/assets` (GET)

The `/assets` API endpoint provides HTTP GET access to asset data.

EXAMPLE

The following example demonstrates a request to the `/assets` API endpoint, resulting in a JSON array that contains asset definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "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
```

```json
1257bb479676bc155b24e21bf93c722b812b0f15cb3bd",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-influxdb-handler",
      "namespace": "default"
    },
    "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"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

API Specification

/assets (GET)

description	Returns the list of 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":
"612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13
a13e7540bac2b63e324f39d9b1257bb479676bc155b24e21bf93c722b81
2b0f15cb3bd",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-influxdb-handler",
      "namespace": "default"
    },
    "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"
      },
      "headers": {
        "Authorization": "Bearer $TOKEN",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    }
  ]
```

`/assets` (POST)

The `/assets` API endpoint provides HTTP POST access to 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`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

API Specification

/assets (POST)	
description	Creates a Sensu 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":
	"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66
	556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eedb706
	4df9019adc8",
	  "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)

The `/assets/:asset` API endpoint

`/assets/:asset` (GET)

The `/assets/:asset` API endpoint provides HTTP GET access to asset data for specific `:asset` definitions, by asset `name`.

EXAMPLE

In the following example, querying the `/assets/:asset` API endpoint returns a JSON map that contains the requested `:asset` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
```

```
[
  {
    "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"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

API Specification

/assets/:asset (GET)	
description	Returns the specified 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":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66
556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eedb706
4df9019adc8",
    "filters": [
      "entity.system.os = 'linux'",
      "entity.system.arch = 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

`/assets/:asset` (PUT)

The `/assets/:asset` API endpoint provides HTTP PUT access to create or update specific `:asset` definitions, by asset name.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/assets/:asset` API endpoint to create the asset `sensu-slack-handler`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

API Specification

/assets/:asset (PUT)	
description	Creates or updates the specified Sensu asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
payload	<code>{ "url": "https://github.com/sensu/sensu-slack-</code>

```
    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'"
      ],
      "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)

`/assets/:asset` (DELETE)

The `/assets/:asset` API endpoint provides HTTP DELETE access so you can delete an asset.

NOTE: Deleting an asset does not remove the downloaded files from the asset cache or remove any references to the deleted asset in other resources.

EXAMPLE

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler \
```

```
  -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 204 No Content
```

API Specification

/assets/:asset (DELETE)	
description	Deletes the specified Sensu 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)

Authentication API

The `/auth` API endpoint

`/auth` (GET)

The `/auth` API endpoint provides HTTP GET access to generate an access token and a refresh token using Sensu's basic authentication.

EXAMPLE

In the following example, querying the `/auth` API endpoint with a given username and password returns an HTTP `200 OK` response to indicate that the credentials are valid, along with an access token and a refresh token.

```
curl -X GET \
http://127.0.0.1:8080/auth \
-u myusername:mypassword

HTTP/1.1 200 OK
{
  "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 see 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)

The `/auth/test` API endpoint

`/auth/test` (GET)

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 provider. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP) or Active Directory (AD).*

In the following example, querying the `/auth/test` API endpoint with a given username and password returns an HTTP `200 OK` response, indicating that the credentials are valid.

```
curl -X GET \
http://127.0.0.1:8080/auth/test \
-u myusername:mypassword


HTTP/1.1 200 OK
```

API Specification

/auth/test (GET)

description	Tests basic authentication credentials (username and password) that were created with Sensu's <u>users API</u>.
example url	<u>http://hostname:8080/auth/test</u>
response codes	¬ **Valid credentials**: 200 (OK) ¬ **Invalid credentials**: 401 (Unauthorized) ¬ **Error**: 500 (Internal Server Error)

The `/auth/token` API endpoint

`/auth/token` (POST)

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 and returns a successful HTTP `200 OK` response along with the new access token.

```
curl -X POST \
http://127.0.0.1:8080/auth/token \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
-H 'Content-Type: application/json' \
-d '{"refresh_token": "eyJhbGciOiJIUzI1NiIs..."}'

HTTP/1.1 200 OK
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1544582187,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
```

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)

Authentication providers API

- The `authproviders` API endpoints (commercial feature)
    - `/authproviders` (GET)
- The `authproviders/:name` API endpoints (commercial feature)
    - `authproviders/:name` (GET)
    - `authproviders/:name` (PUT)
    - `authproviders/:name` (DELETE)

COMMERCIAL FEATURE: Access authentication providers in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

The `/authproviders` API endpoints

`/authproviders` (GET)

The `/authproviders` API endpoint provides HTTP GET access to authentication provider configuration in Sensu.

EXAMPLE

In the following example, querying the `/authproviders` API endpoint returns the authentication provider configuration in Sensu, with an HTTP `200 OK` response.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/authentication/v2/authproviders \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "type": "ldap",
```

```json
    "api_version": "authentication/v2",
    "metadata": {
      "name": "openldap"
    },
    "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
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters. See the API overview for details.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	(see code block below)

```json
[
  {
    "type": "ldap",
    "api_version": "authentication/v2",
    "metadata": {
      "name": "openldap"
    },
    "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": ""
          }
        }
      ]
```

The `/authproviders/:name` API endpoints

`/authproviders/:name` (GET)

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, resulting in an HTTP `200 OK` response.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/authentication/v2/authproviders/openldap \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json'
```

```
HTTP/1.1 200 OK
-d '{
  "type": "ldap",
  "api_version": "authentication/v2",
  "metadata": {
    "name": "openldap"
  },
  "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"
  },
  "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": ""
    }
  }
```

`/authproviders/:name` (PUT)

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, resulting in an HTTP `200 OK` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 200 OK
```

API Specification

/authproviders/:name (PUT)	
description	Creates or updates the authentication provider configuration for the specified name. See the <u>authentication guide</u> for more information about supported providers.
example url	<u>http://hostname:8080/api/enterprise/authentication/v2/authproviders/openldap</u>
payload	```{ "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"
    }
  }
```

payload parameters	All attributes shown in the example payload are required. For more information about configuring authentication providers, see the authentication guide.
response codes	⌐ **Success**: 200 (OK) ⌐ **Malformed**: 400 (Bad Request) ⌐ **Error**: 500 (Internal Server Error)

`/authproviders/:name` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/authproviders/openldap


HTTP/1.1 204 No Content
```

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)

Checks API

The `/checks` API endpoint

`/checks` (GET)

The `/checks` API endpoint provides HTTP GET access to check data.

EXAMPLE

The following example demonstrates a request to the `/checks` API endpoint, resulting in a JSON array that contains check definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks \
```

```
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 200 OK
[
  {
    "command": "check-email.sh -w 75 -c 90",
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": null,
    "subscriptions": [
      "linux"
    ],
    "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-email",
      "namespace": "default"
    }
  }
]
```

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	

```json
[
  {
    "command": "check-email.sh -w 75 -c 90",
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": null,
    "subscriptions": [
      "linux"
    ],
    "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-email",
      "namespace": "default"
    }
  }
```

```
      }
  ]
```

`/checks` (POST)

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 and returns a successful HTTP `200 OK` response and the created check definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "command": "check-cpu.sh -w 75 -c 90",
  "subscriptions": [
    "linux"
  ],
  "interval": 60,
  "publish": true,
  "handlers": [
    "slack"
  ],
  "metadata": {
    "name": "check-cpu",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks

HTTP/1.1 201 Created
```

API Specification

/checks (POST)	
description	Creates a Sensu check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks
| example payload | ```json
{
  "command": "check-cpu.sh -w 75 -c 90",
  "subscriptions": [
    "linux"
  ],
  "interval": 60,
  "publish": true,
  "handlers": [
    "slack"
  ],
  "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, see the check reference. |
| response codes | ⌐ **Success**: 201 (Created)
⌐ **Malformed**: 400 (Bad Request)
⌐ **Error**: 500 (Internal Server Error) |

The `/checks/:check` API endpoint

`/checks/:check` (GET)

The `/checks/:check` API endpoint provides HTTP GET access to check data for `:check` definitions, specified by check name.

EXAMPLE

In the following example, querying the `/checks/:check` API endpoint returns a JSON map that contains the requested `:check` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "command": "check-cpu.sh -w 75 -c 90",
  "handlers": [
    "slack"
  ],
  "high_flap_threshold": 0,
  "interval": 60,
  "low_flap_threshold": 0,
  "publish": true,
  "runtime_assets": null,
  "subscriptions": [
    "linux"
  ],
  "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"
  }
```

```
    }
```

API Specification

<table>
<tr><td colspan="2">

/checks/:check
(GET)

</td></tr>
<tr><td>description</td><td>Returns the specified check.</td></tr>
<tr><td>example url</td><td><u>http://hostname:8080/api/core/v2/namespaces/default/checks/check-cpu</u></td></tr>
<tr><td>response type</td><td>Map</td></tr>
<tr><td>response codes</td><td>

¬ **Success**: 200 (OK)

¬ **Missing**: 404 (Not Found)

¬ **Error**: 500 (Internal Server Error)

</td></tr>
<tr><td>output</td><td>

```
{
  "command": "check-cpu.sh -w 75 -c 90",
  "handlers": [
    "slack"
  ],
  "high_flap_threshold": 0,
  "interval": 60,
  "low_flap_threshold": 0,
  "publish": true,
  "runtime_assets": null,
  "subscriptions": [
    "linux"
  ],
  "proxy_entity_name": "",
  "check_hooks": null,
  "stdin": false,
  "subdue": null,
  "ttl": 0,
  "timeout": 0,
  "round_robin": false,
```

</td></tr>
</table>

```
        "output_metric_format": "",
        "output_metric_handlers": null,
        "env_vars": null,
        "metadata": {
          "name": "check-cpu",
          "namespace": "default"
        }
      }
```

`/checks/:check` (PUT)

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, resulting in an HTTP `200 OK` response and the updated check definition.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "command": "check-cpu.sh -w 75 -c 90",
  "handlers": [
    "slack"
  ],
  "interval": 60,
  "publish": true,
  "subscriptions": [
    "linux"
  ],
  "metadata": {
    "name": "check-cpu",
    "namespace": "default"
  }
}' \
```

```
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu

HTTP/1.1 201 Created
```

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.sh -w 75 -c 90", "handlers": ["slack"], "interval": 60, "publish": true, "subscriptions": ["linux"], "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, see the check reference.
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

`/checks/:check` (DELETE)

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`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu


HTTP/1.1 204 No Content
```

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)

The `/checks/:check/execute` API endpoint

`/checks/:check/execute` (POST)

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 and returns a successful HTTP `202 Accepted` response and an `issued` timestamp.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 202 Accepted
{"issued":1543861798}
```

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.*

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-

---

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)

The `/checks/:check/hooks/:type` API endpoint

`/checks/:check/hooks/:type` (PUT)

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, resulting in a successful HTTP `204 No Content` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

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)

The `/checks/:check/hooks/:type/hook/:hook` API endpoint

`/checks/:check/hooks/:type/hook/:hook` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-
cpu/hooks/critical/hook/process_tree

HTTP/1.1 204 No Content
```

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). See 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)

Cluster API

The `/cluster/members` API endpoint

`/cluster/members` (GET)

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, resulting in a JSON map that contains a Sensu cluster definition.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/cluster/members \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \

HTTP/1.1 200 OK
{
  "header": {
    "cluster_id": 4255616304056076734,
    "member_id": 9882886658148554927,
```

```json
      "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
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
example output	```json { "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"
                      ]
                    }
                  ]
                }
```

`/cluster/members` (POST)

The `/cluster/members` API endpoint provides HTTP POST access to create a Sensu cluster member.

EXAMPLE

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/cluster/members?peer-addrs=http://127.0.0.1:2380

HTTP/1.1 200 OK
{
  "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)

The `/cluster/members/:member` API endpoint

`/cluster/members/:member` (PUT)

The `/cluster/members/:member` API endpoint provides HTTP PUT access to create or update a cluster member, by cluster member ID.

EXAMPLE

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/cluster/members/8927110dc66458af?peer-
addrs=http://127.0.0.1:2380

HTTP/1.1 200 OK
{
  "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: `8927110dc66458af` (hex-encoded uint64 cluster member ID generated using `sensuctl cluster member-list`).

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)

`/cluster/members/:member` (DELETE)

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`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/cluster/members/8927110dc66458a
f


HTTP/1.1 204 No Content
```

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

- `8927110dc66458af` (required): Required hex-encoded uint64 cluster member ID generated using `sensuctl cluster member-list`

---

response codes

- **Success**: 204 (No Content)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

The `/cluster/id` API endpoint

`/cluster/id` (GET)

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, resulting in a string that contains the Sensu cluster ID.

```
curl -X GET \
  -H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/cluster/id

HTTP/1.1 200 OK
"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
response type	String
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
example output	`"23481e76-5844-4d07-b714-6e2ffbbf9315"`

Cluster role bindings API

- ¬ The `/clusterrolebindings` API endpoint
    - ¬ `/clusterrolebindings` (GET)
    - ¬ `/clusterrolebindings` (POST)
- ¬ The `/clusterrolebindings/:clusterrolebinding` API endpoint
    - ¬ `/clusterrolebindings/:clusterrolebinding` (GET)
    - ¬ `/clusterrolebindings/:clusterrolebinding` (PUT)
    - ¬ `/clusterrolebindings/:clusterrolebinding` (DELETE)

The `/clusterrolebindings` API endpoint

`/clusterrolebindings` (GET)

The `/clusterrolebindings` API endpoint provides HTTP GET access to cluster role binding data.

EXAMPLE

The following example demonstrates a request to the `/clusterrolebindings` API endpoint, resulting in a JSON array that contains cluster role binding definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "cluster-admins"
```

```
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-admin"
    },
    "metadata": {
      "name": "cluster-admin"
    }
  },
  {
    "subjects": [
      {
        "type": "Group",
        "name": "system:agents"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "system:agent"
    },
    "metadata": {
      "name": "system:agent"
    }
  }
]
```

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	<pre>[
	  {
	    "subjects": [
	      {
	        "type": "Group",
	        "name": "cluster-admins"
	      }
	    ],
	    "role_ref": {
	      "type": "ClusterRole",
	      "name": "cluster-admin"
	    },
	    "metadata": {
	      "name": "cluster-admin"
	    }
	  },
	  {
	    "subjects": [
	      {
	        "type": "Group",
	        "name": "system:agents"
	      }
	    ],
	    "role_ref": {
	      "type": "ClusterRole",
	      "name": "system:agent"
	    },
	    "metadata": {
	      "name": "system:agent"
	    }
	  }
	]</pre>

`/clusterrolebindings` (POST)

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 and returns a successful HTTP `200 OK` response and the created cluster role binding definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

API Specification

/clusterrolebinding s (POST)

description	Creates a Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings

payload	<pre>{
	  "subjects": [
	    {
	      "type": "User",
	      "name": "bob"
	    }
	  ],
	  "role_ref": {
	    "type": "ClusterRole",
	    "name": "cluster-admin"
	  },
	  "metadata": {
	    "name": "bob-binder"
	  }
	}</pre>

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

The `/clusterrolebindings/:clusterrolebinding` API endpoint

`/clusterrolebindings/:clusterrolebinding` (GET)

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP GET access to cluster role binding data for specific `:clusterrolebinding` definitions, by cluster role binding `name`.

EXAMPLE

In the following example, querying the `/clusterrolebindings/:clusterrolebinding` API endpoint returns a JSON map that contains the requested `:clusterrolebinding` definition (in this example, for the `:clusterrolebinding` named `bob-binder`).

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/bob-binder \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}
```

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

```json
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}
```

`/clusterrolebindings/:clusterrolebinding` (PUT)

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 and returns a successful HTTP `200 OK` response and the created cluster role binding definition.

```
curl -X PUT \
```

```
  -H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
  -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


HTTP/1.1 201 Created
```

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	<pre>{
	  "subjects": [
	    {
	      "type": "Group",
	      "name": "ops"
	    }
	  ],
	  "role_ref": {</pre>

```
      "type": "ClusterRole",
      "name": "cluster-admin"
    },
    "metadata": {
      "name": "ops-group-binder"
    }
  }
}
```

response codes

> ¬ **Success**: 201 (Created)
>
> ¬ **Malformed**: 400 (Bad Request)
>
> ¬ **Error**: 500 (Internal Server Error)

`/clusterrolebindings/:clusterrolebinding` (DELETE)

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 cluster role binding `ops-binding`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-binding


HTTP/1.1 204 No Content
```

API Specification

/clusterrolebinding
s/:clusterrolebindin

g (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)

Cluster roles API

- ¬ The `/clusterroles` API endpoint
    - ¬ `/clusterroles` (GET)
    - ¬ `/clusterroles` (POST)
- ¬ The `/clusterroles/:clusterrole` API endpoint
    - ¬ `/clusterroles/:clusterrole` (GET)
    - ¬ `/clusterroles/:clusterrole` (PUT)
    - ¬ `/clusterroles/:clusterrole` (DELETE)

The `/clusterroles` API endpoint

`/clusterroles` (GET)

The `/clusterroles` API endpoint provides HTTP GET access to cluster role data.

EXAMPLE

The following example demonstrates a request to the `/clusterroles` API endpoint, resulting in a JSON array that contains cluster role definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterroles \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "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"
    }
  }
]
```

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 | ```json
[
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
``` |

```
      ],
      "metadata": {
        "name": "cluster-admin"
      }
    }
  ]
```

`/clusterroles` (POST)

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 and returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

API Specification

/clusterroles (POST)	
description	Creates a Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles

payload	<pre>{
	  "metadata": {
	    "name": "global-event-reader"
	  },
	  "rules": [
	    {
	      "verbs": [
	        "get",
	        "list"
	      ],
	      "resources": [
	        "events"
	      ],
	      "resource_names": null
	    }
	  ]
	}</pre>

response codes

     ¬ **Success**: 201 (Created)

     ¬ **Malformed**: 400 (Bad Request)

     ¬ **Error**: 500 (Internal Server Error)

The `/clusterroles/:clusterrole` API endpoint

`/clusterroles/:clusterrole` (GET)

The `/clusterroles/:clusterrole` API endpoint provides HTTP GET access to cluster role data for specific `:clusterrole` definitions, by cluster role `name`.

EXAMPLE

In the following example, querying the `/clusterroles/:clusterrole` API endpoint returns a JSON map that contains the requested `:clusterrole` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "global-event-reader"
  },
  "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"`
	  `},`
	  `"rules": [`
	    `{`
	      `"verbs": [`
	        `"get",`
	        `"list"`
	      `],`
	      `"resources": [`
	        `"events"`
	      `],`
	      `"resource_names": null`
	    `}`
	  `]`
	`}`

`/clusterroles/:clusterrole` (PUT)

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 The request includes the cluster role definition in the request body and returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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
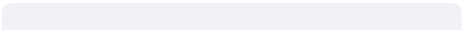
HTTP/1.1 201 Created
```

API Specification

/clusterroles/:clust

errole (PUT)

description	Creates or updates the specified Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
payload	

```json
{
  "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)

`/clusterroles/:clusterrole` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-reader


HTTP/1.1 204 No Content
```

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)

Datastore API

The `/provider` API endpoint

`/provider` (GET)

The `/provider` API endpoint provides HTTP GET access to Sensu datastore data.

EXAMPLE

The following example demonstrates a 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: Bearer $SENSU_ACCESS_TOKEN" \

HTTP/1.1 200 OK
[
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {},
    "spec": {
      "dsn": "postgresql://user:secret@host:port/otherdbname",
```

```
      "pool_size": 20
    }
  },
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {},
    "spec": {
      "dsn": "postgresql://user:secret@host:port/dbname",
      "pool_size": 20
    }
  }
]
```

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	<pre>[
	  {
	    "type": "PostgresConfig",
	    "api_version": "store/v1",
	    "metadata": {},
	    "spec": {
	      "dsn":
	"postgresql://user:secret@host:port/otherdbname",
	      "pool_size": 20
	    }</pre>

```
      },
      {
        "type": "PostgresConfig",
        "api_version": "store/v1",
        "metadata": {},
        "spec": {
          "dsn": "postgresql://user:secret@host:port/dbname",
          "pool_size": 20
        }
      }
    ]
```

The `/provider/:provider` API endpoint

`/provider/:provider` (GET)

EXAMPLE

```
curl -X GET \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres

HTTP/1.1 200 OK
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "dsn": "postgresql://user:secret@host:port/dbname",
    "pool_size": 20
  }
}
```

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 | ```json
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "dsn": "postgresql://user:secret@host:port/dbname",
    "pool_size": 20
  }
}
``` |

`/provider/:provider` (PUT)

EXAMPLE

```
curl -X PUT \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
```

```
-d '{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "dsn": "postgresql://user:secret@host:port/dbname",
    "pool_size": 20
  }
}'


HTTP/1.1 200 OK
```

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": {
"dsn": "postgresql://user:secret@host:port/dbname",
"pool_size": 20
}
}```
response codes	

- ᒣ **Success**: 200 (OK)
- ᒣ **Missing**: 404 (Not Found)
- ᒣ **Error**: 500 (Internal Server Error)

`/provider/:provider` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres


HTTP/1.1 204 No Content
```

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	 ᒣ **Success**: 204 (No Content) ᒣ **Missing**: 404 (Not Found)

Entities API

The `/entities` API endpoint

`/entities` (GET)

The `/entities` API endpoint provides HTTP GET access to entity data.

EXAMPLE

The following example demonstrates a request to the `/entities` API endpoint, resulting in a JSON array that contains the entity definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "entity_class": "agent",
    "sensu_agent_version": "1.0.0",
    "system": {
      "hostname": "sensu-centos",
```

```json
      "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"
    },
    "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",
      "labels": null,
      "annotations": null
    }
  }
]
```

API Specification

<table>
<tr><td colspan="2">/entities (GET)</td></tr>
<tr><td>description</td><td>Returns the list of entities.</td></tr>
<tr><td>example url</td><td><u>http://hostname:8080/api/core/v2/namespaces/default/entities</u></td></tr>
<tr><td>pagination</td><td>This endpoint supports <u>pagination</u> using the <code>limit</code> and <code>continue</code> query parameters.</td></tr>
<tr><td>response filtering</td><td>This endpoint supports <u>API response filtering</u>.</td></tr>
<tr><td>response type</td><td>Array</td></tr>
<tr><td>response codes</td><td>¬ Success: 200 (OK)

¬ Error: 500 (Internal Server Error)</td></tr>
<tr><td>output</td><td>

```
[
  {
```
</td></tr>
</table>

```json
      "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"
      },
      "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",
                "labels": null,
                "annotations": null
              }
            }
          ]
```

`/entities` (POST)

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 and returns a successful `HTTP 201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

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",
	  "subscriptions": [
	    "web"
	  ],
	  "deregister": false,
	  "deregistration": {},
	  "metadata": {
	    "name": "sensu-centos",
	    "namespace": "default",
	    "labels": null,
	    "annotations": null
	  }
	}</pre>

response codes

- ¬ **Success**: 200 (OK)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

The `/entities/:entity` API endpoint

`/entities/:entity` (GET)

The `/entities/:entity` API endpoint provides HTTP GET access to entity data for specific `:entity` definitions, by entity `name`.

EXAMPLE

In the following example, querying the `/entities/:entity` API endpoint returns a JSON map that contains the requested `:entity` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "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": [
```

```json
      {
        "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"
},
"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",
    "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	<pre>{
	  "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": {</pre>

```json
      "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"
  },
  "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",
            "labels": null,
            "annotations": null
        }
    }
```

`/entities/:entity` (PUT)

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 and returns a successful `HTTP 201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


  HTTP/1.1 201 Created
```

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	<pre>{
	  "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
	  }
	}</pre>

response codes	

- ⌐ **Success**: 201 (Created)
- ⌐ **Malformed**: 400 (Bad Request)
- ⌐ **Error**: 500 (Internal Server Error)

`/entities/:entity` (DELETE)

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 `server1`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/server1 \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 204 No Content
```

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/server1
response codes	⌐ **Success**: 204 (No Content) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

Events API

The `/events` API endpoint

`/events` (GET)

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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
```

```json
{
  "timestamp": 1542667666,
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "webserver01",
      "...": "...",
      "arch": "amd64"
    },
    "subscriptions": [
      "testing",
      "entity:webserver01"
    ],
    "metadata": {
      "name": "check-nginx",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  },
  "check": {
    "check_hooks": null,
    "duration": 2.033888684,
    "command": "http_check.sh http://localhost:80",
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 20,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [],
    "subscriptions": [
      "testing"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "ttl": 0,
    "timeout": 0,
    "duration": 0.010849143,
    "output": "",
```

```
      "state": "failing",
      "status": 1,
      "total_state_change": 0,
      "last_ok": 0,
      "occurrences": 1,
      "occurrences_watermark": 1,
      "output_metric_format": "",
      "output_metric_handlers": [],
      "env_vars": null,
      "metadata": {
        "name": "check-nginx",
        "namespace": "default",
        "labels": null,
        "annotations": null
      }
    }
  }
]
```

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	```[```

```json
{
  "timestamp": 1542667666,
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "webserver01",
      "...": "...",
      "arch": "amd64"
    },
    "subscriptions": [
      "testing",
      "entity:webserver01"
    ],
    "metadata": {
      "name": "check-nginx",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  },
  "check": {
    "check_hooks": null,
    "duration": 2.033888684,
    "command": "http_check.sh http://localhost:80",
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 20,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [],
    "subscriptions": [
      "testing"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "ttl": 0,
    "timeout": 0,
    "duration": 0.010849143,
    "output": "",
```

```
                      "state": "failing",
                      "status": 1,
                      "total_state_change": 0,
                      "last_ok": 0,
                      "occurrences": 1,
                      "occurrences_watermark": 1,
                      "output_metric_format": "",
                      "output_metric_handlers": [],
                      "env_vars": null,
                      "metadata": {
                        "name": "check-nginx",
                        "namespace": "default",
                        "labels": null,
                        "annotations": null
                      }
                    }
                  }
                }
              ]
```

`/events` (POST)

The `/events` API endpoint provides HTTP POST access to create an event and send it to the Sensu 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 and returns a successful HTTP `200 OK` response and the event definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json'
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
```

```
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "state": "failing",
    "status": 2,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events

HTTP/1.1 200 OK
```

```
{"timestamp":1552582569,"entity":{"entity_class":"proxy","system":{"network":{"inter
faces":null}},"subscriptions":null,"last_seen":0,"deregister":false,"deregistration"
:{},"metadata":{"name":"server1","namespace":"default"}},"check":{"handlers":["slack
"],"high_flap_threshold":0,"interval":60,"low_flap_threshold":0,"publish":false,"runti
me_assets":null,"subscriptions":[],"proxy_entity_name":"","check_hooks":null,"stdin"
:false,"subdue":null,"ttl":0,"timeout":0,"round_robin":false,"executed":0,"history":
null,"issued":0,"output":"Server
error","state":"failing","status":2,"total_state_change":0,"last_ok":0,"occurrences"
:0,"occurrences_watermark":0,"output_metric_format":"","output_metric_handlers":null
,"env_vars":null,"metadata":{"name":"server-health"}},"metadata":{}}
```

API Specification

/events (POST)	
description	Creates a new Sensu event. To update an existing event, use the /events/:entity/:check PUT endpoint.
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,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}
```

response codes

        ¬ **Success**: 200 (OK)

        ¬ **Malformed**: 400 (Bad Request)

        ¬ **Error**: 500 (Internal Server Error)

The `/events/:entity` API endpoint

`/events/:entity` (GET)

The `/events/:entity` API endpoint provides HTTP GET access to event data specific to an `:entity`, by entity `name`.

EXAMPLE

In the following example, querying the `/events/:entity` API endpoint returns a list of Sensu events for the `sensu-go-sandbox` entity and a successful HTTP `200 OK` response.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "timestamp": 1543871497,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "linux",
        "entity:sensu-go-sandbox"
      ],
      "last_seen": 1543858763,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      }
    },
    "check": {
      "command": "check-cpu.sh -w 75 -c 90",
      "duration": 1.054253257,
      "executed": 1543871496,
      "history": [
        {
          "status": 0,
          "executed": 1543870296
        }
      ],
      "issued": 1543871496,
      "output": "CPU OK - Usage:.50\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1543871497,
      "occurrences": 1,
```

```json
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      }
    },
    "metadata": {
      "namespace": "default"
    }
  },
  {
    "timestamp": 1543871524,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "linux",
        "entity:sensu-go-sandbox"
      ],
      "last_seen": 1543871523,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      }
    },
    "check": {
      "handlers": [
        "keepalive"
      ],
      "executed": 1543871524,
      "history": [
        {
          "status": 0,
          "executed": 1543871124
        }
      ],
      "issued": 1543871524,
      "output": "",
      "state": "passing",
```

```
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1543871524,
      "occurrences": 1,
      "metadata": {
        "name": "keepalive",
        "namespace": "default"
      }
    },
    "metadata": {}
  }
 ]
```

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/sensu-go-sandbox
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	``` [{ "timestamp": 1543871524, "entity": { "entity_class": "agent", ```

```json
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "linux",
        "entity:sensu-go-sandbox"
      ],
      "last_seen": 1543871523,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      }
    },
    "check": {
      "handlers": [
        "keepalive"
      ],
      "executed": 1543871524,
      "history": [
        {
          "status": 0,
          "executed": 1543871124
        }
      ],
      "issued": 1543871524,
      "output": "",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1543871524,
      "occurrences": 1,
      "metadata": {
        "name": "keepalive",
        "namespace": "default"
      }
    },
    "metadata": {}
  }
]
```

The `/events/:entity/:check` API endpoint

`/events/:entity/:check` (GET)

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 `server-health` check.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 200 OK


{
    "timestamp": 1577724113,
    "entity": {
        "entity_class": "proxy",
        "system": {
            "network": {
                "interfaces": null
            }
        },
        "subscriptions": null,
        "last_seen": 0,
        "deregister": false,
        "deregistration": {},
        "metadata": {
            "name": "server1",
            "namespace": "default"
        },
        "sensu_agent_version": ""
```

```json
        },
        "check": {
            "handlers": [
                "slack"
            ],
            "high_flap_threshold": 0,
            "interval": 60,
            "low_flap_threshold": 0,
            "publish": false,
            "runtime_assets": null,
            "subscriptions": [],
            "proxy_entity_name": "",
            "check_hooks": null,
            "stdin": false,
            "subdue": null,
            "ttl": 0,
            "timeout": 0,
            "round_robin": false,
            "executed": 1543880280,
            "history": [
                {
                    "status": 1,
                    "executed": 1543880296
                },
                {
                    "status": 2,
                    "executed": 1543880435
                },
                {
                    "status": 1,
                    "executed": 1543889363
                }
            ],
            "issued": 0,
            "output": "Server error",
            "state": "failing",
            "status": 1,
            "total_state_change": 0,
            "last_ok": 0,
            "occurrences": 1,
            "occurrences_watermark": 1,
            "output_metric_format": "",
```

```
        "output_metric_handlers": null,
        "env_vars": null,
        "metadata": {
            "name": "server-health",
            "namespace": "default"
        }
    },
    "metadata": {}
}
```

The request returns an HTTP `200 OK` response and the resulting event definition.

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/server-health
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output	<pre>{
	    "timestamp": 1577724113,
	    "entity": {
	        "entity_class": "proxy",
	        "system": {
	            "network": {
	                "interfaces": null
	            }
	        },</pre>

```json
            "subscriptions": null,
            "last_seen": 0,
            "deregister": false,
            "deregistration": {},
            "metadata": {
                "name": "server1",
                "namespace": "default"
            },
            "sensu_agent_version": ""
        },
        "check": {
            "handlers": [
                "slack"
            ],
            "high_flap_threshold": 0,
            "interval": 60,
            "low_flap_threshold": 0,
            "publish": false,
            "runtime_assets": null,
            "subscriptions": [],
            "proxy_entity_name": "",
            "check_hooks": null,
            "stdin": false,
            "subdue": null,
            "ttl": 0,
            "timeout": 0,
            "round_robin": false,
            "executed": 1543880280,
            "history": [
                {
                    "status": 1,
                    "executed": 1543880296
                },
                {
                    "status": 2,
                    "executed": 1543880435
                },
                {
                    "status": 1,
                    "executed": 1543889363
                }
            ],
```

```
            "issued": 0,
            "output": "Server error",
            "state": "failing",
            "status": 1,
            "total_state_change": 0,
            "last_ok": 0,
            "occurrences": 1,
            "occurrences_watermark": 1,
            "output_metric_format": "",
            "output_metric_handlers": null,
            "env_vars": null,
            "metadata": {
                "name": "server-health",
                "namespace": "default"
            }
        },
        "metadata": {}
    }
```

`/events/:entity/:check` (POST)

The `/events/:entity/:check` API endpoint provides HTTP POST access to create or update an event and send it to the Sensu 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 `slack` event handler. The event includes a status code of `1`, indicating a warning, and an output message of `Server error`.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json'
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
```

```
      "name": "server1",

      "namespace": "default"

    }

  },

  "check": {

    "output": "Server error",

    "status": 1,

    "handlers": ["slack"],

    "interval": 60,

    "metadata": {

      "name": "server-health"

    }

  }

}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The request returns an HTTP `200 OK` response and the resulting event definition.

NOTE: A namespace is not required to create the event. The event will use the namespace in the URL by default.

```
HTTP/1.1 200 OK
{"timestamp":1552582569,"entity":{"entity_class":"proxy","system":{"network":{"inter
faces":null}},"subscriptions":null,"last_seen":0,"deregister":false,"deregistration"
:{},"metadata":{"name":"server1","namespace":"default"}},"check":{"handlers":["slack
"],"high_flap_threshold":0,"interval":60,"low_flap_threshold":0,"publish":false,"runti
me_assets":null,"subscriptions":[],"proxy_entity_name":"","check_hooks":null,"stdin"
:false,"subdue":null,"ttl":0,"timeout":0,"round_robin":false,"executed":0,"history":
null,"issued":0,"output":"Server
error","status":1,"total_state_change":0,"last_ok":0,"occurrences":0,"occurrences_wa
termark":0,"output_metric_format":"","output_metric_handlers":null,"env_vars":null,"
metadata":{"name":"server-health"}},"metadata":{}}
```

You can use sensuctl or the Sensu dashboard to see the event:

```
sensuctl event list
```

You should see 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 (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 | ```json
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}
``` |
| response codes | |

- ¬ **Success**: 200 (OK)
- ¬ **Missing**: 404 (Not Found)
- ¬ **Error**: 500 (Internal Server Error)

`/events/:entity/:check` (PUT)

The `/events/:entity/:check` API endpoint provides HTTP PUT access to create or update an event and send it to the Sensu 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 `slack` event handler. The event includes a status code of `1`, indicating a warning, and an output message of `Server error`.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json'
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}' \
```

```
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The request returns an HTTP `200 OK` response and the resulting event definition.

NOTE: A namespace is not required to create the event. The event will use the namespace in the URL by default.

```
HTTP/1.1 200 OK
{"timestamp":1552582569,"entity":{"entity_class":"proxy","system":{"network":{"inter
faces":null}},"subscriptions":null,"last_seen":0,"deregister":false,"deregistration"
:{},"metadata":{"name":"server1","namespace":"default"}},"check":{"handlers":["slack
"],"high_flap_threshold":0,"interval":60,"low_flap_threshold":0,"publish":false,"runti
me_assets":null,"subscriptions":[],"proxy_entity_name":"","check_hooks":null,"stdin"
:false,"subdue":null,"ttl":0,"timeout":0,"round_robin":false,"executed":0,"history":
null,"issued":0,"output":"Server
error","status":1,"total_state_change":0,"last_ok":0,"occurrences":0,"occurrences_wa
termark":0,"output_metric_format":"","output_metric_handlers":null,"env_vars":null,"
metadata":{"name":"server-health"}},"metadata":{}}
```

You can use sensuctl or the Sensu dashboard to see the event:

```
sensuctl event list
```

You should see 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 | ```json
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}
``` |
| payload parameters | See the payload parameters section below. |
| response codes | ⌐ **Success**: 200 (OK)

⌐ **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, see 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, see the check specification.

Example request with minimum required event attributes

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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 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 event `handlers` . For more information about these attributes and their available values, see the event specification.

Example request with minimum recommended event attributes

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
```

```
  -H 'Content-Type: application/json'
  -d '{
    "entity": {
      "entity_class": "proxy",
      "metadata": {
        "name": "server1",
        "namespace": "default"
      }
    },
    "check": {
      "output": "Server error",
      "status": 1,
      "handlers": ["slack"],
      "interval": 60,
      "metadata": {
        "name": "server-health"
      }
    }
  }' \
  http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

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. See the <u>events reference</u> and for more information about Sensu metric format.

Example request including metrics

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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
      }
    ]
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-metrics
```

`/events/:entity/:check` (DELETE)

EXAMPLE

The following example shows a request to the `/events/:entity/:check` API endpoint to delete the event produced by the `sensu-go-sandbox` entity and `check-cpu` check, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox/check-
cpu \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 204 No Content
```

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/events/sensu-go-sandbox/check-cpu
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Federation API

- The `/etcd-replicators` endpoint
    - `/etcd-replicators` (GET)
    - `/etcd-replicators` (POST)
- The `/etcd-replicators/:etcd-replicator` endpoint
    - `/etcd-replicators/:etcd-replicator` (GET)
    - `/etcd-replicators/:etcd-replicator` (PUT)
    - `/etcd-replicators/:etcd-replicator` (DELETE)
- The `/clusters` endpoint
    - `/clusters` (GET)
- The `/clusters/:cluster` endpoint
    - `/clusters/:cluster` (GET)
    - `/clusters/:cluster` (PUT)
    - `/clusters/:cluster` (DELETE)

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

The `/etcd-replicators` endpoint

NOTE: The etcd-replicators datatype is only accessible for users who have a cluster role that permits access to replication resources.

`/etcd-replicators` (GET)

The `/etcd-replicators` API endpoint provides HTTP GET access to a list of replicators.

EXAMPLE

The following example demonstrates a request to the `/etcd-replicators` API endpoint, resulting in a list of replicators.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
[
  {
    "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
    }
  }
]
```

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

```machine_data
[
  {
    "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
    }
  }
]
```

`/etcd-replicators` (POST)

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: Bearer $SENSU_ACCESS_TOKEN" \
  -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

  HTTP/1.1 200 OK
```

API Specification

/etcd-replicators (POST)	
description	Creates a new replicator (if none exists). **NOTE**: *If you do not specify a namespace when you create a replicator, all namespaces for the given resource are replicated.*
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)

The `/etcd-replicators/:etcd-replicator` API endpoint

NOTE: The etcd-replicators datatype is only accessible for users who have a cluster role that permits access to replication resources.

`/etcd-replicators/:etcd-replicator` (GET)

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP GET access to data for a specific `:etcd-replicator`, by replicator name.

EXAMPLE

In the following example, querying the `/etcd-replicators/:etcd-replicator` API endpoint returns a JSON map that contains the requested `:etcd-replicator`.

```
  curl -X GET \
```

```
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
{
  "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
  }
}
```

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"
  },
  "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
  }
}
```

`/etcd-replicators/:etcd-replicator` (PUT)

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: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 200 OK
```

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	<pre>{
	  "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",</pre>

```
        "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)

`/etcd-replicators/:etcd-replicator` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator

HTTP/1.1 204 No Content
```

API Specification

/etcd-replicators/:etcd-replicator

(DELETE)	
description	Deletes the specified replicator from Sensu.
example url	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

The `/clusters` endpoint

`/clusters` (GET)

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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK

[
    {
        "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"
            ]
        }
    }
 ]
```

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"
},
"spec": {
"api_urls": [
"http://10.0.0.1:8080",
"http://10.0.0.2:8080",
"http://10.0.0.3:8080"
]
}
}
``

```
      }
  ]
```

The `/clusters/:cluster` endpoint

`/clusters/:cluster` (GET)

The `/clusters/:cluster` API endpoint provides HTTP GET access to data for a specific `cluster`, by cluster name.

EXAMPLE

In the following example, querying the `/clusters/:cluster` API endpoint returns a JSON map that contains the requested `:etcd-replicator`.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 200 OK

{
  "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"
      ]
  }
}
```

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 | ```json
{
    "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"
        ]
    }
}
``` |

`/clusters/:cluster` (PUT)

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: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 200 OK
```

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	

```json
{
    "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)

`/clusters/:cluster` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a
```

```
HTTP/1.1 204 No Content
```

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)

Filters API

The `/filters` API endpoint

`/filters` (GET)

The `/filters` API endpoint provides HTTP GET access to event filter data.

EXAMPLE

The following example demonstrates a request to the `/filters` API endpoint, resulting in a JSON array that contains event filter definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters \
-H "Authorization: Bearer $TOKEN"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "development_filter",
      "namespace": "default"
    },
```

```
      "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	

```json
[
  {
    "metadata": {
      "name": "development_filter",
      "namespace": "default"
    },
    "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
  }
]
```

`/filters` (POST)

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`. The request returns a successful HTTP `201 Created` response

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


  HTTP/1.1 201 Created
```

API Specification

/filters (POST)	
description	Creates a Sensu event filter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/filters
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)

The `/filters/:filter` API endpoint

`/filters/:filter` (GET)

The `/filters/:filter` API endpoint provides HTTP GET access to event filter data for specific `:filter` definitions, by filter name.

EXAMPLE

In the following example, querying the `/filters/:filter` API endpoint returns a JSON map that contains the requested `:filter` definition (in this example, 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"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "state_change_only",
    "namespace": "default"
  },
  "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 | ```json
{
  "metadata": {
    "name": "state_change_only",
    "namespace": "default"
  },
  "action": "allow",
  "expressions": [
    "event.check.occurrences == 1"
  ],
  "runtime_assets": null
}
``` |

`/filters/:filter` (PUT)

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` .The request returns a successful HTTP `200 OK` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

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)

`/filters/:filter` (DELETE)

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`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/development_filter \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 204 No Content
```

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)

Handlers API

The `/handlers` API endpoint

`/handlers` (GET)

The `/handlers` API endpoint provides HTTP GET access to handler data.

EXAMPLE

The following example demonstrates a request to the `/handlers` API endpoint, resulting in a JSON array that contains handler definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "influx-db",
      "namespace": "default"
    },
```

```
    "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/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	

```json
[
  {
    "metadata": {
      "name": "influx-db",
      "namespace": "default"
    },
    "type": "pipe",
    "command": "sensu-influxdb-handler -d sensu",
    "timeout": 0,
    "handlers": null,
    "filters": null,
    "env_vars": [

"INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:808
6",
      "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"]
      }
    ]
```

`/handlers` (POST)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

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:808
6",
    "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)

The `/handlers/:handler` API endpoint

`/handlers/:handler` (GET)

The `/handlers/:handler` API endpoint provides HTTP GET access to handler data for specific `:handler` definitions, by handler `name`.

EXAMPLE

In the following example, querying the `/handlers/:handler` API endpoint returns a JSON map that contains the requested `:handler` definition (in this example, for the `:handler` named `slack`).

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/slack \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "slack",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-slack-handler --channel '#monitoring'",
```

```
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXX
XXXXXXXXXXXX"
    ],
    "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", "labels": null, "annotations": null }, ```

```
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T000000
00/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
```

`/handlers/:handler` (PUT)

The `/handlers/:handler` API endpoint provides HTTP GET 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-dbdevelopment_filter`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

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)

`/handlers/:handler` (DELETE)

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`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/slack \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 204 No Content
```

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)

Health API

The `/health` API endpoint

`/health` (GET)

The `/health` API endpoint provides HTTP GET access to health data for your Sensu instance.

EXAMPLE

The following example demonstrates a request to the `/health` API endpoint, resulting in a JSON map that contains Sensu health data.

```
curl -X GET \
http://127.0.0.1:8080/health

HTTP/1.1 200 OK
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 2882886652148554927,
      "MemberIDHex": "8923110df66458af",
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 4255616344056076734,
    "member_id": 2882886652148554927,
    "raft_term": 26
  }
}
```

API Specification

/health (GET)	
description	Returns health information about the Sensu instance.
example url	http://hostname:8080/health
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output	<pre>{
	  "Alarms": null,
	  "ClusterHealth": [
	    {
	      "MemberID": 2882886652148554927,
	      "MemberIDHex": "8923110df66458af",
	      "Name": "default",
	      "Err": "",
	      "Healthy": true
	    }
	  ],
	  "Header": {
	    "cluster_id": 4255616344056076734,
	    "member_id": 2882886652148554927,
	    "raft_term": 26
	  }
	}</pre>

Hooks API

The `/hooks` API endpoint

`/hooks` (GET)

The `/hooks` API endpoint provides HTTP GET access to hook data.

EXAMPLE

The following example demonstrates a request to the `/hooks` API endpoint, resulting in a JSON array that contains hook definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "nginx-log",
      "namespace": "default"
    },
```

```json
    "command": "tail -n 100 /var/log/nginx/error.log",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  },
  {
    "metadata": {
      "name": "process-tree",
      "namespace": "default"
    },
    "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"
          },
          "command": "tail -n 100 /var/log/nginx/error.log",
          "timeout": 10,
          "stdin": false,
          "runtime_assets": null
      },
      {
          "metadata": {
            "name": "process-tree",
            "namespace": "default"
          },
          "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu |
      head -n 6",
          "timeout": 10,
          "stdin": false,
          "runtime_assets": null
      }
  ]
```

`/hooks` (POST)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

API Specification

/hooks (POST)	
description	Creates a Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks

payload	<code>{
	  "metadata": {
	    "name": "process-tree",
	    "namespace": "default",
	    "labels": null,
	    "annotations": null
	  },
	  "command": "ps aux",
	  "timeout": 10,
	  "stdin": false
	}</code>

response codes

    ¬ **Success**: 201 (Created)

    ¬ **Malformed**: 400 (Bad Request)

    ¬ **Error**: 500 (Internal Server Error)

The `/hooks/:hook` API endpoint

`/hooks/:hook` (GET)

The `/hooks/:hook` API endpoint provides HTTP GET access to hook data for specific `:hook` definitions, by hook name.

EXAMPLE

In the following example, querying the `/hooks/:hook` API endpoint returns a JSON map that contains the requested `:hook` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "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 | ```json
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "ps aux",
  "timeout": 10,
  "stdin": false
}
``` |

`/hooks/:hook` (PUT)

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` . The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

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)

`/hooks/:hook` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/process-tree \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 204 No Content
```

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)

License management API

The `/license` API endpoints

For more information about commercial features designed for enterprises, see Get started with commercial features.

`/license` (GET)

The `/license` API endpoint provides HTTP GET access to the active license configuration.

EXAMPLE

The following example demonstrates a request to the `/license` API endpoint, resulting in a JSON array that contains the license definition.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/license \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json'

HTTP/1.1 200 OK
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {},
  "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	```
	{
	``

```json
        "type": "LicenseFile",
        "api_version": "licensing/v2",
        "metadata": {},
        "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": {}
        }
      }
```

`/license` (PUT)

The `/license` API endpoint provides HTTP PUT access to activate a commercial license.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/license` API endpoint to create the license definition. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
```

```
-d '{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {},
  "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/core/v2/namespaces/default/license

HTTP/1.1 201 Created
```

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

```json
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {},
  "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)

`/license` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/enterprise/licensing/v2/license \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 204 No Content
```

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 API

The `/metrics` API endpoint

`/metrics` (GET)

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 request to the `/metrics` API endpoint, resulting in plaintext output that contains internal Sensu metrics.

```
curl -X GET \
http://127.0.0.1:8080/metrics

HTTP/1.1 200 OK
# 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_millisecon
ds Bucketed histogram of db compaction pause duration.
# TYPE
etcd_debugging_mvcc_db_compaction_pause_duration_millisecon
ds histogram
etcd_debugging_mvcc_db_compaction_pause_duration_millisecon
ds_bucket{le="1"} 0
etcd_debugging_mvcc_db_compaction_pause_duration_millisecon
```

```
ds_bucket{le="2"} 0
...
```

Mutators API

The `/mutators` API endpoint

`/mutators` (GET)

The `/mutators` API endpoint provides HTTP GET access to mutator data.

EXAMPLE

The following example demonstrates a request to the `/mutators` API endpoint, resulting in a JSON array that contains mutator definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "default",
      "labels": null,
```

```
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
  }
]
```

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", "labels": null, "annotations": null }, "command": "example_mutator.go", "timeout": 0,```

```
        "env_vars": [],
        "runtime_assets": []
    }
  ]
```

`/mutators` (POST)

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` .The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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": []
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators

HTTP/1.1 201 Created
```

API Specification

/mutators (POST)	
description	Creates a Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators

payload	<pre>{
	  "metadata": {
	    "name": "example-mutator",
	    "namespace": "default",
	    "labels": null,
	    "annotations": null
	  },
	  "command": "example_mutator.go",
	  "timeout": 0,
	  "env_vars": [],
	  "runtime_assets": []
	}</pre>

response codes	
	¬ **Success**: 201 (Created)
	¬ **Malformed**: 400 (Bad Request)
	¬ **Error**: 500 (Internal Server Error)

The `/mutators/:mutator` API endpoint

`/mutators/:mutator` (GET)

The `/mutators/:mutator` API endpoint provides HTTP GET access to mutator data for specific `:mutator` definitions, by mutator name.

EXAMPLE

In the following example, querying the `/mutators/:mutator` API endpoint returns a JSON map that

contains the requested `:mutator` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
```

API Specification

/mutators/:mutator (GET)	
description	Returns the specified mutator.
example url	http://hostname:8080/api/core/v2/namespaces/default/mutators/mutator-name
response type	Map
response codes	˥ **Success**: 200 (OK) ˥ **Missing**: 404 (Not Found) ˥ **Error**: 500 (Internal Server Error)

output

```json
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
```

`/mutators/:mutator` (PUT)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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": []
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator


HTTP/1.1 201 Created
```

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	<pre>{
	  "metadata": {
	    "name": "example-mutator",
	    "namespace": "default",
	    "labels": null,
	    "annotations": null
	  },
	  "command": "example_mutator.go",
	  "timeout": 0,
	  "env_vars": [],
	  "runtime_assets": []
	}</pre>

response codes

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

`/mutators/:mutator` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \


HTTP/1.1 204 No Content
```

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)

Namespaces API

The `/namespaces` API endpoint

`/namespaces` (GET)

The `/namespaces` API endpoint provides HTTP GET access to namespace data.

EXAMPLE

The following example demonstrates a request to the `/namespaces` API endpoint, resulting in a JSON array that contains namespace definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "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` query parameter.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)

output	```[
	  {
	    "name": "default"
	  },
	  {
	    "name": "development"
	  }
	]```

`/namespaces` (POST)

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` . The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/namespaces


HTTP/1.1 201 Created
```

API Specification

/namespaces (POST)	
description	Creates a Sensu namespace.
example URL	http://hostname:8080/api/core/v2/namespaces
payload	```{
"name": "development"
}```
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

The `/namespaces/:namespace` API endpoint

`/namespaces/:namespace` (PUT)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/namespaces/development

HTTP/1.1 201 Created
```

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)

`/namespaces/:namespace` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/development \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 204 No Content
```

API Specification

/namespaces/:na mespace (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)

The `/user-namespaces` API endpoint

`/user-namespaces` (GET)

The `/user-namespaces` API endpoint provides HTTP GET access to the namespaces the user has access to.

EXAMPLE

The following example demonstrates a request to the `/user-namespaces` API endpoint, resulting in a JSON array that contains the namespaces the user has access to.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/user-namespaces \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "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"
  }
]
``` |

Role bindings API

The `/rolebindings` API endpoint

`/rolebindings` (GET)

The `/rolebindings` API endpoint provides HTTP GET access to role binding data.

EXAMPLE

The following example demonstrates a request to the `/rolebindings` API endpoint, resulting in a JSON array that contains role binding definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "readers"
```

```
      }
    ],
    "role_ref": {
      "type": "Role",
      "name": "read-only"
    },
    "metadata": {
      "name": "readers-group-binding",
      "namespace": "default"
    }
  }
]
```

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"
      }
    }
  ]
```

`/rolebindings` (POST)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings


HTTP/1.1 201 Created
```

API Specification

/rolebindings (POST)	
description	Creates a Sensu role binding.
example URL	http://hostname:8080/api/core/v2/namespaces/default/rolebindings
| payload | ```json
{
  "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) |

The `/rolebindings/:rolebinding` API endpoint

`/rolebindings/:rolebinding` (GET)

The `/rolebindings/:rolebinding` API endpoint provides HTTP GET access to role binding data for specific `:rolebinding` definitions, by role binding `name`.

EXAMPLE

In the following example, querying the `/rolebindings/:rolebinding` API endpoint returns a JSON map that contains the requested `:rolebinding` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default"
  }
}
```

API Specification

/rolebindings/:role binding (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

```json
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default"
  }
}
```

`/rolebindings/:rolebinding` (PUT)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

API Specification

/rolebindings/:role binding (PUT)	
description	Creates or updates a Sensu role binding.
example URL	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/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)

`/rolebindings/:rolebinding` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 204 No Content
```

API Specification

/rolebindings/:role binding (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)

Roles API

The `/roles` API endpoint

`/roles` (GET)

The `/roles` API endpoint provides HTTP GET access to role data.

EXAMPLE

The following example demonstrates a request to the `/roles` API endpoint, resulting in a JSON array that contains role definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "rules": [
      {
        "verbs": [
          "get",
```

```json
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "event-reader",
    "namespace": "default"
  }
},
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}
]
```

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"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "read"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
``` |

```
          ],
          "metadata": {
            "name": "read-only",
            "namespace": "default"
          }
        }
      ]
```

`/roles` (POST)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

API Specification

/roles (POST)	
description	Creates a Sensu role.
example URL	http://hostname:8080/api/core/v2/namespaces/default/roles

payload	
	``json
	{
	  "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)

The `/roles/:role` API endpoint

`/roles/:role` (GET)

The `/roles/:role` API endpoint provides HTTP GET access to role data for specific `:role` definitions, by role name.

EXAMPLE

In the following example, querying the `/roles/:role` API endpoint returns a JSON map that contains the requested `:role` definition (in this example, 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: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}
```

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	<pre>{
	  "rules": [
	    {
	      "verbs": [
	        "read"
	      ],
	      "resources": [
	        "*"
	      ],
	      "resource_names": null
	    }
	  ],
	  "metadata": {
	    "name": "read-only",
	    "namespace": "default"
	  }
	}</pre>

`/roles/:role` (PUT)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

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

```json
{
  "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)

`/roles/:role` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only \
```

```
  -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"


HTTP/1.1 204 No Content
```

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)

Silencing API

The `/silenced` API endpoint

`/silenced` (GET)

The `/silenced` API endpoint provides HTTP GET access to silencing entry data.

EXAMPLE

The following example demonstrates a request to the `/silenced` API endpoint, resulting in a JSON array that contains silencing entry definitions.

```
curl -X GET \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced


HTTP/1.1 200 OK
```

```json
[
  {
    "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
  }
]
```

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	```json
	[
	  {
	    "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
      }
    ]
```

`/silenced` (POST)

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`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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


HTTP/1.1 201 Created
```

API Specification

/silenced (POST)	
description	Creates a Sensu silencing entry.
example URL	http://hostname:8080/api/core/v2/namespaces/default/silenced

payload	<pre>{
	  "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
	}</pre>

response codes	⌐ **Success**: 201 (Created) ⌐ **Malformed**: 400 (Bad Request) ⌐ **Error**: 500 (Internal Server Error)

The `/silenced/:silenced` API endpoint

`/silenced/:silenced` (GET)

The `/silenced/:silenced` API endpoint provides HTTP GET access to silencing entry data for specific `:silenced` definitions, by silencing entry name.

EXAMPLE

In the following example, querying the `/silenced/:silenced` API endpoint returns a JSON map that contains the requested silencing entry definition (in this example, for the silencing entry named `linux:check-cpu`).Silencing entry names are generated from the combination of a subscription name and check name.

```
curl -X GET \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu

HTTP/1.1 200 OK
{
  "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
}
```

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	```json\n{\n "metadata": {\n "name": "linux:check-cpu",\n "namespace": "default",\n "labels": null,\n "annotations": null\n },\n "expire": -1,\n "expire_on_resolve": false,\n "creator": "admin",\n "reason": "reason for silence",\n "subscription": "linux",\n "begin": 1542671205\n}\n```

`/silenced/:silenced` (PUT)

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` . The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-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

HTTP/1.1 201 Created
```

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)

`/silenced/:silenced` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu


HTTP/1.1 204 No Content
```

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)

The `/silenced/subscriptions/:subscription` API endpoint

`/silenced/subscriptions/:subscription` (GET)

The `/silenced/subscriptions/:subscription` API endpoint provides HTTP GET access to silencing entry data by subscription name.

EXAMPLE

In the following example, querying the `silenced/subscriptions/:subscription` API endpoint returns a JSON array that contains the requested silences for the given subscription (in this example, for the `linux` subscription).

```
curl -X GET \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/subscriptions/linux

HTTP/1.1 200 OK
[
  {
    "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
  }
]
```

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",
        "labels": null,
        "annotations": null
      },
      "expire": -1,
      "expire_on_resolve": false,
      "creator": "admin",
      "reason": "reason for silence",
      "subscription": "linux",
      "begin": 1542671205
    }
  ]
```

The `/silenced/checks/:check` API endpoint

`/silenced/checks/:check` (GET)

The `/silenced/checks/:check` API endpoint provides HTTP GET access to silencing entry data by check name.

EXAMPLE

In the following example, querying the `silenced/checks/:check` API endpoint returns a JSON array that contains the requested silences for the given check (in this example, for the `check-cpu` check).

```
curl -X GET \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/checks/check-cpu

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
```

```
      "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	<pre>[
	  {
	    "metadata": {
	      "name": "linux:check-cpu",
	      "namespace": "default",
	      "labels": null,</pre>

```json
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "check": "linux",
    "begin": 1542671205
  }
]
```

Tessen API

The `/tessen` API endpoints

The Tessen API provides HTTP access to manage Tessen configuration. Access to the Tessen API is restricted to the default `admin` user.

`/tessen` (GET)

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. The request returns an HTTP `200 OK` response and a JSON map that contains the active Tessen configuration, indicating whether Tessen is enabled.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/tessen \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "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 }```

`/tessen` (PUT)

The `/tessen` API endpoint provides HTTP PUT access to opt in to or opt out of Tessen. Tessen is enabled by default on Sensu backends and required for licensed Sensu instances.

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. The request returns an HTTP `200 OK` response and the resulting Tessen configuration.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "opt_out": false
}' \
http://127.0.0.1:8080/api/core/v2/tessen
```

```
HTTP/1.1 200 OK
{
  "opt_out": false
}
```

API Specification

/tessen (PUT)	
description	Updates the active Tessen configuration. Licensed Sensu instances override the `opt_out` attribute to `false` at runtime.
example url	http://hostname:8080/api/core/v2/tessen
request parameters	Required: `opt_out` (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`
	`}`

Users API

NOTE: *The users API allows you to create and manage user credentials with Sensu's built-in basic authentication provider. To configure user credentials with an external provider like Lightweight Directory Access Protocol (LDAP) or Active Directory (AD), use Sensu's authentication providers API.*

The `/users` API endpoint

`/users` (GET)

The `/users` API endpoint provides HTTP GET access to user data.

EXAMPLE

The following example demonstrates a request to the `/users` API, resulting in a JSON array that contains user definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/users \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
[
  {
    "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
}
]```

`/users` (POST)

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`, resulting in an HTTP `201 Created` response and the created user definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
```

```
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "temporary",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users

HTTP/1.1 201 Created
```

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	```\n{\n "username": "alice",\n "groups": [\n "ops"\n],\n "password": "temporary",\n "disabled": false\n}\n```
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request)

The `/users/:user` API endpoint

`/users/:user` (GET)

The `/users/:user` API endpoint provides HTTP GET access to <u>user data</u> for a specific user by `username`.

EXAMPLE

In the following example, querying the `/users/:user` API returns a JSON map that contains the requested <u>:user</u> <u>definition</u> (in this example, for the `alice` user).

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/users/alice \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN"

HTTP/1.1 200 OK
{
  "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 | ```json
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
``` |

`/users/:user` (PUT)

The `/users/:user` API endpoint provides HTTP PUT access to create or update <u>user data</u> for a specific user by `username`.

EXAMPLE

The following example demonstrates a PUT request to the `/users` API endpoint to update the user `alice` (in this case, to reset the user's password), resulting in an HTTP `201 Created` response and the updated user definition.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "reset-password",
```

```
   "disabled": false
 }' \
 http://127.0.0.1:8080/api/core/v2/users/alice


 HTTP/1.1 201 Created
```

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"
	  ],
	  "password": "reset-password",
	  "disabled": false
	}
	``

response codes

       ¬ **Success**: 201 (Created)

       ¬ **Malformed**: 400 (Bad Request)

       ¬ **Error**: 500 (Internal Server Error)

`/users/:user` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice


HTTP/1.1 204 No Content
```

NOTE: This endpoint **disables** but does not delete the user. You can <u>reinstate</u> 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)

The `/users/:user/password` API endpoint

`/users/:user/password` (PUT)

The `/users/:user/password` API endpoint provides HTTP PUT access to update a user's 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`, resulting in an HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "password": "newpassword"
}' \
http://127.0.0.1:8080/api/core/v2/users/alice/password


HTTP/1.1 201 Created
```

API Specification

/users/:user/password (PUT)	
description	Updates 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) and `password` (string; the user's new password).
payload	```{ "username": "admin", "password": "newpassword" }```
response codes	⌐ **Success**: 201 (Created) ⌐ **Malformed**: 400 (Bad Request) ⌐ **Error**: 500 (Internal Server Error)

The `/users/:user/reinstate` API endpoint

`/users/:user/reinstate` (PUT)

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`, resulting in an HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
http://127.0.0.1:8080/api/core/v2/users/alice/reinstate


HTTP/1.1 201 Created
```

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)

The `/users/:user/groups` API endpoint

`/users/:user/groups` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups

HTTP/1.1 204 No Content
```

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)

The `/users/:user/groups/:group` API endpoints

`/users/:user/groups/:group` (PUT)

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`, resulting in a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops


HTTP/1.1 201 Created
```

API Specification

/users/:user/group s/: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)

`/users/:user/groups/:group` (DELETE)

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 `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops


HTTP/1.1 204 No Content
```

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)

Version API

The `/version` API endpoint

`/version` (GET)

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 request to the `/version` API endpoint, resulting in a JSON map that contains Sensu version data.

```
curl -X GET \
http://127.0.0.1:8080/version

HTTP/1.1 200 OK
{
  "etcd": {
    "etcdserver": "3.3.17",
    "etcdcluster": "3.3.0"
  },
  "sensu_backend": "5.x.x#yyyyyyy"
}
```

API Specification

/version (GET)	
description	Returns the Sensu backend and etcd version for the Sensu instance.

example url	http://hostname:8080/version
response type	Map
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)
response parameters	Required: ⌐ `etcd.etcdserver` (string). Etcd server version. ⌐ `sensu_backend` (string). Sensu backend version in the format x.x.x#yyyyyyy where x.x.x is the Sensu version and yyyyyyy is the release SHA Optional: ⌐ `etcd.etcdcluster` (string). Etcd cluster version for Sensu instances with the default embedded etcd. Not required to match the etcd server version or the cluster versions of other backends in the cluster.
output	

```
{
  "etcd": {
    "etcdserver": "3.3.17",
    "etcdcluster": "3.3.0"
  },
  "sensu_backend": "5.x.x#yyyyyyy"
}
```

Sensuctl quickstart

Configure sensuctl and log in

```
sensuctl configure
? Sensu Backend URL: http://127.0.0.1:8080
? Username: YOUR_USERNAME
? Password: YOUR_PASSWORD
```

NOTE: *If you are using Docker, the default username is* `admin` *and the password is* `P@ssw0rd!` *.*

Create resources from a file that contains JSON resource definitions

```
sensuctl create --file filename.json
```

View monitored entities

```
sensuctl entity list
```

View monitoring events

```
sensuctl event list
```

Edit a check

In this example, the check name is `check-cpu` :

```
sensuctl edit check check-cpu
```

View the JSON configuration for a check

In this example, the check name is `check-cpu`:

```
sensuctl check info check-cpu --format wrapped-json
```

Sensuctl

Sensuctl is a command line tool for managing resources within Sensu.It works by calling Sensu's underlying API to create, read, update, and delete resources, events, and entities.Sensuctl is available for Linux, macOS, and Windows.See Install Sensu to install and configure sensuctl.

First-time setup

To set up sensuctl, run `sensuctl configure` to log in to sensuctl and connect to the Sensu backend:

```
sensuctl configure
```

When prompted, type the Sensu backend URL and your Sensu access credentials.

```
? Sensu Backend URL: http://127.0.0.1:8080
? Username: YOUR_USERNAME
? Password: YOUR_PASSWORD
? Namespace: default
? Preferred output format: tabular
```

NOTE: *If you are using Docker, the default username is* `admin` *and the password is* `P@ssw0rd!` .

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, see the backend reference.

Username, Password, and Namespace

When you install the Sensu backend, during the initialization step, you create a username and password for a `default` namespace. 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, see the RBAC reference.

NOTE: *If you are using Docker, the* `sensu-backend init` *command for initialization runs automatically with a default username (* `admin` *) and password (* `P@ssw0rd!` *) for Docker. You do not need to create a username and password for the* `default` *namespace if you are using Docker.*

Preferred output format

Sensuctl supports the following output formats:

    ¬ `tabular` : A user-friendly, columnar format

- ⌐ `wrapped-json` : An accepted format for use with `sensuctl create`
- ⌐ `yaml` : An accepted format for use with `sensuctl create`
- ⌐ `json` : A format used by the Sensu API

After you are logged in, you can change the output format with `sensuctl config set-format` or set the output format per command with the `--format` flag.

Non-interactive mode

Run `sensuctl configure` non-interactively by adding the `-n` (`--non-interactive`) flag.

```
sensuctl configure -n --url http://127.0.0.1:8080 --username YOUR_USERNAME --password
YOUR_PASSWORD --format tabular
```

NOTE: *If you are using Docker, the default username is* `admin` *and the password is* `P@ssw0rd!` *.*

Get help

Sensuctl supports a `--help` flag for each command and subcommand.

See command and global flags

```
sensuctl --help
```

See subcommands and flags

```
sensuctl check --help
```

See usage and flags

```
sensuctl check delete --help
```

Manage sensuctl

The `sencutl config` command lets you view the current sensuctl configuration and set the namespace and output format.

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 output format

Use the `set-format` command to change the default output format for the current user.

For example, to change the output format to `tabular`:

```
sensuctl config set-format tabular
```

Set namespace

Use the `set-namespace` command to change the default namespace for the current user. For more information about configuring Sensu access control, see 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
```

Global flags

Global flags modify settings specific to sensuctl, such as the Sensu backend URL and namespace. You can use global flags with most sensuctl commands.

```
--api-url string            host URL of Sensu installation
--cache-dir string          path to directory containing cache & temporary files
--config-dir string         path to directory containing configuration files
--insecure-skip-tls-verify  skip TLS certificate verification (not recommended!)
--namespace string          namespace in which we perform actions
--trusted-ca-file string    TLS CA certificate bundle in PEM format
```

You can set these flags permanently by editing `.config/sensu/sensuctl/{cluster, profile}`.

Test a user password

To test the password for a user created with Sensu's built-in <u>basic authentication</u>:

```
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 <u>basic authentication provider</u>. It does not test user credentials defined via an authentication provider like <u>Lightweight Directory Access Protocol (LDAP)</u> or <u>Active Directory (AD)</u>.

For example, if you test LDAP credentials with the `sensuctl user test-creds` command, the backend will log an error, even if you know 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"}
```

Create resources

The `sensuctl create` command allows you to create or update resources by reading from STDIN or a flag configured file (`-f`). The `create` command accepts Sensu resource definitions in `wrapped-json` and `yaml`. Both JSON and YAML resource definitions wrap the contents of the resource in

`spec` and identify the resource `type` .See the `wrapped-json` example and this table for a list of supported types.See the reference docs for information about creating resource definitions.

`wrapped-json` format

In this example, the file `my-resources.json` specifies two resources: a `marketing-site` check and a `slack` handler, separated *without* a comma:

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata" : {
    "name": "marketing-site",
    "namespace": "default"
  },
  "spec": {
    "command": "check-http.rb -u https://sensu.io",
    "subscriptions": ["demo"],
    "interval": 15,
    "handlers": ["slack"]
  }
}
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

 "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXXX"
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
```

```
      "handlers": [],
      "runtime_assets": [],
      "timeout": 0,
      "type": "pipe"
  }
}
```

To create all resources from `my-resources.json` using `sensuctl create`:

```
sensuctl create --file my-resources.json
```

Or:

```
cat my-resources.json | sensuctl create
```

`yaml` format

In this example, the file `my-resources.yml` specifies two resources: a `marketing-site` check and a `slack` handler, separated with three dashes (`---`).

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: marketing-site
  namespace: default
spec:
  command: check-http.rb -u https://sensu.io
  subscriptions:
  - demo
  interval: 15
  handlers:
  - slack
---
type: Handler
```

```
api_version: core/v2
metadata:
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXX
XXXXXXXXXXX
  filters:
  - is_incident
  - not_silenced
  type: pipe
```

To create all resources from `my-resources.yml` using `sensuctl create` :

```
sensuctl create --file my-resources.yml
```

Or:

```
cat my-resources.yml | sensuctl create
```

sensuctl create resource types

sensuctl create types			
AdhocRequest	adhoc_request	Asset	asset
CheckConfig	check_config	ClusterRole	cluster_role
ClusterRoleBinding	cluster_role_binding	Entity	entity
EtcdReplicators	Event	event	EventFilter

event_filter	Handler	handler	Hook
hook	HookConfig	hook_config	Mutator
mutator	Namespace	namespace	Role
role	RoleBinding	role_binding	Silenced
silenced	ldap	ad	TessenConfig
PostgresConfig			

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 and namespaced resource types, see the RBAC reference.

The `sensuctl create` command applies namespaces to resources in the following order, from highest precedence to lowest:

1. **Namespaces 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. See Manage sensuctl to view your current session config and set the session namespace.

In this example, the file `pagerduty.yml` defines a handler *without* a `namespace` attribute:

```
type: Handler
api_version: core/v2
metadata:
  name: pagerduty
spec:
  command: sensu-pagerduty-handler
```

```
    env_vars:
    - PAGERDUTY_TOKEN=SECRET
    type: pipe
```

To create the `pagerduty` handler in the `default` namespace:

```
sensuctl create --file pagerduty.yml --namespace default
```

To create the `pagerduty` handler in the `production` namespace:

```
sensuctl create --file pagerduty.yml --namespace production
```

To create the `pagerduty` handler in the current session namespace:

```
sensuctl create --file pagerduty.yml
```

Delete resources

The `sensuctl delete` command allows you to delete resources by reading from STDIN or a flag configured file (`-f`). The `delete` command accepts Sensu resource definitions in `wrapped-json` and `yaml` formats and uses the same resource types as `sensuctl create` . To be deleted successfully, resources provided to the `delete` command must match the name and namespace of an existing resource.

To delete all resources from `my-resources.yml` with `sensuctl delete` :

```
sensuctl delete --file my-resources.yml
```

Or:

```
cat my-resources.yml | sensuctl delete
```

Delete resources across namespaces

If you omit the `namespace` attribute from resource definitions, you can use the `senusctl delete --namespace` flag to specify the namespace for a group of resources at the time of deletion. This allows you to remove resources across namespaces without manual editing. See the Create resources across namespaces section for usage examples.

Update resources

Sensuctl allows you to update resource definitions with a text editor. To use `sensuctl edit`, specify the resource type and resource name.

For example, to edit a handler named `slack` with `sensuctl edit`:

```
sensuctl edit handler slack
```

sensuctl edit resource types

sensuctl edit types			
`asset`	`check`	`cluster`	`cluster-role`
`cluster-role-binding`	`entity`	`event`	`filter`
`handler`	`hook`	`mutator`	`namespace`
`role`	`role-binding`	`silenced`	`user`
`auth`			

Export resources

The `sensuctl dump` command allows you to export your resources to standard out or to a file.You can export all of your resources or a subset of them based on a list of resource types.The `dump` command supports exporting in `wrapped-json` and `yaml`.

NOTE: Passwords are not included when exporting users. You must add the `password` *attribute to any exported user resources before they can be used with* `sensuctl create`.

To export all resources to a file named `my-resources.yaml` in `yaml` format:

```
sensuctl dump all --format yaml --file my-resources.yaml
```

To export only checks to standard out in `yaml` format:

```
sensuctl dump check --format yaml
```

To export only handlers and filters to a file named `my-handlers-and-filters.yaml` in `yaml` format:

```
sensuctl dump handler,filter --format yaml --file my-handlers-and-filters.yaml
```

sensuctl dump resource types

You can use the sensuctl `--types` subcommand to list the types of supported resources:

```
sensuctl dump --types
```

The table below lists supported `sensuctl dump` resource types.

NOTE: The resource types with no synonym listed are <u>commercial features</u>.

Synonym	Fully qualified name
None	`authentication/v2.Provider`

None	`licensing/v2.LicenseFile`
None	`store/v1.PostgresConfig`
None	`federation/v1.Replicator`
`assets`	`core/v2.Asset`
`checks`	`core/v2.CheckConfig`
`clusterroles`	`core/v2.ClusterRole`
`clusterrolebindings`	`core/v2.ClusterRoleBinding`
`entities`	`core/v2.Entity`
`events`	`core/v2.Event`
`filters`	`core/v2.EventFilter`
`handlers`	`core/v2.Handler`
`hooks`	`core/v2.Hook`
`mutators`	`core/v2.Mutator`
`namespaces`	`core/v2.Namespace`
`roles`	`core/v2.Role`
`rolebindings`	`core/v2.RoleBinding`
`silenced`	`core/v2.Silenced`
`tessen`	`core/v2.TessenConfig`
`users`	`core/v2.User`

Manage resources

Sensuctl provides the following commands to manage Sensu resources.

- ⌐ `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 role`
- ⌐ `sensuctl role-binding`
- ⌐ `sensuctl silenced`
- ⌐ `sensuctl tessen`
- ⌐ `sensuctl user`

Subcommands

Sensuctl provides a standard set of list, info, and delete operations for most resource types.

```
list                        list resources
info NAME                   show detailed resource information given resource name
delete NAME                 delete resource given resource name
```

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.json` in `wrapped-json` format:

```
sensuctl check list --format wrapped-json > my-resources.json
```

To see the definition for a check named `check-cpu` in `wrapped-json` format:

```
sensuctl check info check-cpu --format wrapped-json
```

In addition to the standard operations, commands may support subcommands or flags that allow you to take special action based on the resource type. The sections below describe these resource-specific operations.

For a list of subcommands specific to a resource, run `sensuctl TYPE --help`.

Handle large datasets

When querying sensuctl for large datasets, use the `--chunk-size` flag with any `list` command to avoid timeouts and improve performance.

For example, the following command returns the same output as `sensuctl event list` but makes multiple API queries (each for the number of objects specified by `--chunk-size`) instead of one API query for the complete dataset:

```
sensuctl event list --chunk-size 500
```

sensuctl check

In addition to the <u>standard subcommands</u>, the `sensuctl check execute` command executes a check on demand, given the check name:

```
sensuctl check execute 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
```

sensuctl cluster

The `sensuctl cluster` command lets you manage a Sensu cluster using the following subcommands:

```
health            get Sensu health status
id                get unique 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 see the health of your Sensu cluster:

```
sensuctl cluster health
```

sensuctl event

In addition to the standard subcommands, you can use `sensuctl event resolve` to manually resolve events:

```
sensuctl event resolve ENTITY CHECK
```

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
```

sensuctl namespace

See the RBAC reference for information about using access control with namespaces.

sensuctl user

See the RBAC reference for information about local user management with sensuctl.

Response filters

COMMERCIAL FEATURE: Access sensuctl response filters in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

Sensuctl supports response filtering for all `list` commands using the `--label-selector` and `--field-selector` flags. For information about the operators and fields you can use in response filters, see the API docs.

Response filter syntax quick reference

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`
`&&`	Logical AND	`check.publish == true && slack in check.handlers`

Filter responses with labels

Use the `--label-selector` flag to filter responses using custom labels.

In this example, the command returns entities with the `proxy_type` label set to `switch`:

```
sensuctl entity list --label-selector 'proxy_type == switch'
```

Filter responses with resource attributes

Use the `--field-selector` flag to filter responses using selected resource attributes. To see the resource attributes you can use in response filter statements, see the API docs.

In this example, the command returns entities with the `switches` subscription:

```
sensuctl entity list --field-selector 'switches in entity.subscriptions'
```

You can also combine the `--label-selector` and `--field-selector` flags.

In this example, the command returns checks with the `region` label set to `us-west-1` that use the `slack` handler:

```
sensuctl check list --label-selector 'region == "us-west-1"' --field-selector 'slack
in check.handlers'
```

Time formats

Sensuctl supports multiple time formats depending on the manipulated resource.Supported canonical time zone IDs are defined in the tz database.

WARNING: *Windows does not support canonical zone IDs (for example,* `America/Vancouver` *).*

Dates with time

Use full dates with time to specify an exact point in time.This is useful for setting silences, for example.

Sensuctl supports the following formats:

- RFC3339 with numeric zone offset: `2018-05-10T07:04:00-08:00` or `2018-05-10T15:04:00Z`
- RFC3339 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`

Shell auto-completion

Installation (Bash shell)

Make sure bash-completion is installed.If you use a current Linux in a non-minimal installation, bash-completion should be available.

On macOS, install with:

```
brew install bash-completion
```

Then add this to your `~/.bash_profile` :

```
if [ -f $(brew --prefix)/etc/bash_completion ]; then
. $(brew --prefix)/etc/bash_completion
fi
```

After bash-completion is installed, add this to your `~/.bash_profile` :

```
source <(sensuctl completion bash)
```

Now you can source your `~/.bash_profile` or launch a new terminal to use shell auto-completion.

```
source ~/.bash_profile
```

Installation (ZSH)

Add this to your `~/.zshrc` :

```
source <(sensuctl completion zsh)
```

Now you can source your `~/.zshrc` or launch a new terminal to use shell auto-completion.

```
source ~/.zshrc
```

Usage

`sensuctl` Tab

```
check        configure   event       user
asset        completion  entity       handler
```

```
sensuctl check  Tab
```

```
create   delete   import   list
```

Environment variables

Sensuctl includes the `sensuctl env` command to help export and set environment variables on your systems.

```
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 (e.g. 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_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
```

Usage
BASH

```bash
sensuctl env
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"


# Run this command to configure your shell:
# eval $(sensuctl env)
```

CMD

```
sensuctl env --shell 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
REM Run this command to configure your shell:
REM  @FOR /f "tokens=*" %i IN ('sensuctl env --shell cmd') DO @%i
```

POWERSHELL

```
sensuctl env --shell 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"


# Run this command to configure your shell:
# & sensuctl env --shell powershell | Invoke-Expression
```

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` .

For example:

```
cat .config/sensu/sensuctl/profile
{
  "format": "tabular",
  "namespace": "demo",
  "username": "admin"
}
```

```
cat .config/sensu/sensuctl/cluster
{
  "api-url": "http://localhost:8080",
  "trusted-ca-file": "",
  "insecure-skip-tls-verify": false,
  "access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "expires_at": 1550082282,
  "refresh_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

These 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.

Interact with Bonsai

Sensuctl supports installing asset definitions directly from Bonsai, the Sensu asset index, and checking your Sensu backend for outdated assets.You can also use `sensuctl command` to install, execute, list, and delete commands from Bonsai or a URL.

Install asset definitions

To install an asset definition directly from Bonsai, use `sensuctl asset add [NAMESPACE/NAME]` `[:VERSION]` . `[:VERSION]` is only required if you require a specific version or are pinning to a specific version.

Replace `[NAMESPACE/NAME]` with the namespace and name of the asset from Bonsai:

sensu/sensu-influxdb-handler (17) Versions 3.1.2 ▾ ⬇ Supported Tier

Sensu Go InfluxDB Metrics Handler

```
sensuctl asset add sensu/sensu-influxdb-handler:3.1.1
fetching bonsai asset: sensu/sensu-influxdb-handler:3.1.1
added asset: sensu/sensu-influxdb-handler:3.1.1
```

You can also use the `--rename` flag to rename the asset on install:

```
sensuctl asset add sensu/sensu-slack-handler --rename slack-handler
no version specified, using latest: 1.0.3
fetching bonsai asset: sensu/sensu-slack-handler:1.0.3
added asset: sensu/sensu-slack-handler:1.0.3
```

Check your Sensu backend for outdated assets

To check your Sensu backend for assets that have newer versions available on Bonsai, use `sensuctl asset outdated`. This will print a list of assets installed in the backend whose version is older than the newest version available on Bonsai:

```
sensuctl asset outdated
         Asset Name                    Bonsai Asset         Current Version  Latest
  Version
```

```
------------------------  ------------------------  --------------  -------
-------
sensu/sensu-influxdb-handler   sensu/sensu-influxdb-handler      3.1.1
3.1.2
```

Extend sensuctl with commands

Use `sensuctl command` to install, execute, list, and delete commands from Bonsai or a URL.

NOTE: To use `sensuctl command` assets, please upgrade to Sensu Go 5.17.0. A bug that prevented Bonsai-installed `sensuctl command` assets from working in 5.16.x is fixed in the 5.17.0 release.

Install commands

To install a sensuctl command from Bonsai or a URL:

```
sensuctl command install [ALIAS] ([NAMESPACE/NAME]:[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 `[NAMESPACE/NAME]` with the name of the asset from Bonsai. `[: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 (e.g. 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 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
8b15a170e091dab42256fe64ca7c4a050ed49a9dbfd6c8129c95506a8a9a91f2762ac1a6d24f4fc54543
0613fd45abc91d3e5d3605fcfffb270dcf01996caa7f
```

NOTE: Asset definitions with multiple asset builds are only supported via Bonsai.

Execute commands

To execute a sensuctl command plugin via its asset's bin/entrypoint executable:

```
sensuctl command exec [ALIAS] [args] [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 `[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.

Replace `[args]` with the globlal flags you want to use.Run `sensuctl command exec -h` to view global flags.To pass `[args]` flags to the bin/entrypoint executable, make sure to specify them after a double dash surrounded by spaces.

NOTE: When you use `sensuctl command exec` *, the environment variables are passed to the command.*

For example:

```
sensuctl command exec mycommand arg1 arg2 --cache-dir /tmp -- --flag1 --flag2=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 arg1 arg2 --flag1 --flag2=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 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 delete -h` to view flags. Flags are optional and apply only to the `delete` command.

Sensu agent

The Sensu agent is a lightweight client that runs on the infrastructure components you want to monitor. Agents register with the Sensu backend as [monitoring 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. See the [installation guide](#) to install the agent.

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.

Create monitoring events using service checks

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 agent subscriptions.

Each Sensu agent has a defined set of `subscriptions` : a list of roles and responsibilities assigned to the system (for example, a webserver or database). These subscriptions determine which monitoring 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. For an agent to execute a service check, you must specify the same subscription in the agent configuration and the check definition.

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 assets and stores them in its local cache. By default, agents cache asset data at `/var/cache/sensu/sensu-agent` (`C:\ProgramData\sensu\cache\sensu-agent` on Windows systems) or as specified by the the `cache-dir` flag.
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.

Subscription configuration

To configure subscriptions for an agent, set the `subscriptions` flag. To configure subscriptions for a check, set the check definition attribute `subscriptions` .

In addition to the subscriptions defined in the agent configuration, Sensu agent entities also 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.

Proxy entities

Sensu proxy entities allow Sensu to monitor external resources on systems or devices where a Sensu agent cannot be installed (such 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 event data in place of the agent entity data. See the entity reference and Monitor external resources for more information about monitoring proxy entities.

Create monitoring 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 by the API configuration flags. Only unsecured HTTP (no HTTPS) is supported at this time. Any requests for unknown endpoints result in an HTTP `404 Not Found` response.

`/events` (POST)

The `/events` API provides HTTP POST access to publish monitoring events to the Sensu backend pipeline via the agent API. The agent places events created via the `/events` POST 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 `/events` API uses a configurable burst limit and rate limit for relaying events to the backend. See API configuration flags to configure the `events-burst-limit` and `events-rate-limit` flags.

Example POST request to events API

The following example submits an HTTP POST request to the `/events` API. The request creates 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.

```
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
```

```
HTTP/1.1 202 Accepted
```

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, see the the check reference.

You can use the Sensu agent API to enable tasks that run outside of Sensu's check scheduling to emit events.Using the check TTL attribute, 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 a silent failure (which might otherwise be missed).If an external source sends a Sensu event with a check TTL to the Sensu agent API, Sensu expects another event from the same external source before the TTL expires.

In this example, external event input via the Sensu agent API 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.

This script sends 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
```

With this initial event submitted to the agent API, you recorded in the Sensu backend that your script started. You also configured the dead man's switch so that 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 monitoring 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, you 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
```

When you omit the TTL attribute from this event, you also remove 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

```json
{
  "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 `/healthz` API 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 `/healthz` API:

```
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 or `sensu backend unavailable` if the agent cannot connect to a backend.
example url	http://hostname:3031/healthz

Create monitoring 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 (TCP on Windows systems) 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 formatted according to the StatsD line protocol:

```
<metricname>:<value>|<type>
```

For more information, see the StatsD documentation.

Configure the StatsD listener

To configure the StatsD listener, specify the `statsd-event-handlers` configuration flag in the agent configuration, and start the agent.

```
# Start an agent that sends StatsD metrics to InfluxDB
sensu-agent --statsd-event-handlers influx-db
```

Use the StatsD configuration flags to change the default settings for the StatsD listener address, port, and flush interval.

```
# Start an agent with a customized address and flush interval
sensu-agent --statsd-event-handlers influx-db --statsd-flush-interval 1 --statsd-
metrics-host 123.4.5.11 --statsd-metrics-port 8125
```

Create monitoring 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 flags.

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	`"output": "CheckHttp OK: 200, 78572 bytes"`

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	`"source": "sensu-docs-site"`

client

description	*NOTE*: The `client` attribute is deprecated in favor of the `source` attribute (see above). 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.
required	false
default	The agent entity that receives the event data.
type	String
example	`"client": "sensu-docs-site"`

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	`"executed": 1458934742`

duration

description	Amount of time it took to execute the check. In seconds.
required	false

type	Float
example	`"duration": 1.903135228`

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	`"command": "check-http.rb -u https://sensuapp.org"`

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	`"interval": 60`

handlers

description	Array of Sensu handler names to use for handling the event. Each

<div style="text-align: center;">handler name in the array must be a string.</div>

required	false
type	Array
example	```"handlers": ["slack", "influxdb"]```

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` flag.

If a Sensu agent fails to send keepalive events over the period specified by the `keepalive-critical-timeout` flag, the Sensu backend creates a keepalive **critical** alert in the Sensu dashboard.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` flag, the Sensu backend creates a keepalive **warning** alert in the Sensu dashboard.The value you specify for `keepalive-warning-timeout` must be lower than the value you specify for `keepalive-critical-timeout`.

You can use keepalives to identify unhealthy systems and network partitions, send notifications, and trigger auto-remediation, among other useful actions.

NOTE: Keepalive monitoring is not supported for proxy entities, as they are inherently unable to run a Sensu agent.

Handle keepalive events

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 pipeline, 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

```yml
type: Handler
api_version: core/v2
metadata:
  name: keepalive
  namespace: default
spec:
  handlers:
  - slack
  type: set
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "keepalive",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack"
    ]
  }
}
```

Service management

Start the service

Use the `sensu-agent` tool to start the agent and apply configuration flags.

Linux

To start the agent with <u>configuration flags</u>:

```
sensu-agent start --subscriptions disk-checks --log-level debug
```

To see available configuration flags and defaults:

```
sensu-agent start --help
```

To start the agent using a service manager:

```
sudo service sensu-agent start
```

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 an admin 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:\\monitoring\\sensu\\config\\agent.yml' --
log-file 'C:\\monitoring\\sensu\\log\\sensu-agent.log'
```

Stop the service

To stop the agent service using a service manager:

Linux

```
sudo service sensu-agent stop
```

Windows

```
sc.exe stop SensuAgent
```

Restart the service

You must restart the agent to implement any configuration updates.

To restart the agent using a service manager:

Linux

```
sudo service sensu-agent restart
```

Windows

```
sc.exe stop SensuAgent
sc.exe start SensuAgent
```

Enable on boot

To enable the agent to start on system boot:

Linux

```
sudo systemctl enable sensu-agent
```

To disable the agent from starting on system boot:

```
sudo systemctl disable sensu-agent
```

NOTE: *On older distributions of Linux, use* `sudo chkconfig sensu-agent on` *to enable the agent and* `sudo chkconfig sensu-agent off` *to disable the agent.*

Windows

The service is configured to start automatically on boot by default.

Get service status

To see the status of the agent service using a service manager:

Linux

```
service sensu-agent status
```

Windows

```
sc.exe query SensuAgent
```

Get service version

To get the current agent version using the `sensu-agent` tool:

```
sensu-agent version
```

Uninstall the service

Windows

```
sensu-agent service uninstall
```

Get help

The `sensu-agent` tool provides general and command-specific help flags:

```
# Show sensu-agent commands
sensu-agent help

# Show options for the sensu-agent start subcommand
sensu-agent start --help
```

Registration

In practice, agent registration happens when a Sensu backend processes an agent keepalive event for an agent that is not already registered in the Sensu agent registry (based on the configured agent `name`). The Sensu backend stores this agent registry, and it is accessible via `sensuctl entity list`.

All Sensu agent data provided in keepalive events gets stored in the agent registry and used to add context to Sensu events and detect Sensu agents in an unhealthy state.

Registration events

If a Sensu event handler named `registration` is configured, the Sensu backend creates and processes an event for agent registration, applying any configured filters and mutators before

executing the configured handler.

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

You can use registration events to execute one-time handlers for new Sensu agents. For example, you can use registration event handlers to update external configuration management databases (CMDBs) such as ServiceNow.

The handlers reference includes an example registration event handler.

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.

Deregistration events

As with registration events, the Sensu backend can create and process a deregistration event when the Sensu agent process stops. You can use deregistration events to trigger a handler that updates external CMDBs or performs an action to update ephemeral infrastructures. To enable deregistration events, use the `deregister` flag, and specify the event handler using the `deregistration-handler` flag. You can specify a deregistration handler per agent using the `deregistration-handler` agent flag or by setting a default for all agents using the `deregistration-handler` backend configuration flag.

Cluster

Agents can connect to a Sensu cluster by specifying any Sensu backend URL in the cluster in the `backend-url` configuration flag. For more information about clustering, see Backend datastore configuration flags and Run a Sensu cluster.

Synchronize time

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.

Configuration

The agent loads configuration upon startup, so you must restart the agent for any configuration updates to take effect.

Linux

Specify the agent configuration with either a `.yml` file or `sensu-agent start` command line flags. Configuration via command line flags overrides attributes specified in a configuration file.See the Example Sensu agent configuration file for flags and defaults.

Configuration summary

```
$ sensu-agent start --help
start the sensu agent

Usage:
  sensu-agent start [flags]

Flags:
    --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)
    --backend-url strings                 ws/wss URL of Sensu backend server (to
specify multiple backends use this flag 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                     TLS certificate in PEM format
  -c, --config-file string                path to sensu-agent config file
    --deregister                           ephemeral agent
    --deregistration-handler string       deregistration handler that should
process the entity deregistration event.
    --disable-assets                      disable check assets on this agent
    --disable-api                         disable the Agent HTTP API
    --disable-sockets                     disable the Agent TCP and UDP event
sockets
    --events-burst-limit                  /events api burst limit
    --events-rate-limit                   maximum number of events transmitted
to the backend through the /events api
```

```
  -h, --help                                    help for start
      --insecure-skip-tls-verify        skip ssl verification
      --keepalive-critical-timeout uint32     number of seconds until agent is
considered dead by backend to create a critical event (default 0)
      --keepalive-interval uint32             number of seconds to send between
keepalive events (default 20)
      --keepalive-warning-timeout uint32      number of seconds until agent is
considered dead by backend to create a warning event (default 120)
      --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] (default "warn")
      --name string                           agent name (defaults to hostname)
(default "my-hostname")
      --namespace string                      agent namespace (default "default")
      --password string                       agent password (default "P@ssw0rd!")
      --redact string                         comma-delimited customized list of
fields to redact
      --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
      --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 string                  comma-delimited list of agent
subscriptions
      --trusted-ca-file string                tls certificate authority
      --user string                           agent user (default "agent")
```

Windows

You can specify the agent configuration using a `.yml` file.See the example agent configuration file

(also provided with Sensu packages at `%ALLUSERSPROFILE%\sensu\config\agent.yml.example` ; default `C:\ProgramData\sensu\config\agent.yml.example`).

General configuration flags

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_ANNOTATIONS`
example	```# Command line examples
sensu-agent start --annotations
sensu.io/plugins/slack/config/webhook-
url=https://hooks.slack.com/services/T00000000/B00000000/XX
XXXXXXXXXXXXXXXXXXXXXX
sensu-agent start --annotations example-key="example value"
--annotations example-key2="example value"

/etc/sensu/agent.yml example
annotations:
  sensu.io/plugins/slack/config/webhook-url:
"https://hooks.slack.com/services/T00000000/B00000000/XXXXX
XXXXXXXXXXXXXXXXXXX"``` |

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`
environment variable	`SENSU_BACKEND_URL`
example	

```
# Command line examples
sensu-agent start --backend-url ws://0.0.0.0:8081
sensu-agent start --backend-url ws://0.0.0.0:8081 --
backend-url ws://0.0.0.0:8082

# /etc/sensu/agent.yml example
backend-url:
  - "ws://0.0.0.0:8081"
  - "ws://0.0.0.0:8082"
```

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`
example	

```
# Command line example
```

```
sensu-agent start --cache-dir /cache/sensu-agent


# /etc/sensu/agent.yml 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`

example	``` # Command line example sensu-agent start --config-file /sensu/agent.yml sensu-agent start -c /sensu/agent.yml # /etc/sensu/agent.yml example config-file: "/sensu/agent.yml" ```

disable-assets

description	When set to `true`, disables assets for the agent. If an agent attempts to execute a check that requires an 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`
| example | ```
# Command line example
sensu-agent start --disable-assets

# /etc/sensu/agent.yml example
disable-assets: true
``` |

allow-list

description	Path to yaml or json file that contains the allow list of check or hook commands the agent can execute. See the example allow list configuration file and the allow list configuration spec for information about building a configuration file.
type	String
default	`""`
environment variable	`SENSU_ALLOW_LIST`
| example | ```
# Command line example
sensu-agent start --allow-list /etc/sensu/check-allow-list.yaml

# /etc/sensu/agent.yml example
allow-list: /etc/sensu/check-allow-list.yaml
``` |

labels

| description | Custom attributes to include with event data that you can use for response and dashboard view filtering and tokens.

If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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`
environment variable	`SENSU_LABELS`

example	<pre># Command line examples
	sensu-agent start --labels proxy_type=website
	sensu-agent start --labels example_key1="example value"
	example_key2="example value"
	
	# /etc/sensu/agent.yml example
	labels:
	  proxy_type: "website"</pre>

name

description	Entity name assigned to the agent entity.
type	String
default	Defaults to hostname (for example, `sensu-centos`).
environment variable	`SENSU_NAME`

example	<pre># Command line example
	sensu-agent start --name agent-01
	
	# /etc/sensu/agent.yml example
	name: "agent-01"</pre>

log-level

description	Logging level: `panic`, `fatal`, `error`, `warn`, `info`, or `debug`.
type	String
default	`warn`
environment variable	`SENSU_LOG_LEVEL`
example	```# Command line example
sensu-agent start --log-level debug

/etc/sensu/agent.yml example
log-level: "debug"``` |

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`
example	```# Command line examples
sensu-agent start --subscriptions disk-checks,process-checks
sensu-agent start --subscriptions disk-checks --subscriptions process-checks

/etc/sensu/agent.yml example
subscriptions:
  - disk-checks
  - process-checks``` |

API configuration flags

api-host

description	Bind address for the Sensu agent HTTP API.
type	String
default	`127.0.0.1`
environment variable	`SENSU_API_HOST`
example	

```
# Command line example
sensu-agent start --api-host 0.0.0.0

# /etc/sensu/agent.yml example
api-host: "0.0.0.0"
```

api-port

description	Listening port for the Sensu agent HTTP API.
type	Integer
default	`3031`
environment variable	`SENSU_API_PORT`
example	

```
# Command line example
sensu-agent start --api-port 4041

# /etc/sensu/agent.yml example
api-port: 4041
```

disable-api

description	`true` to disable the agent HTTP API. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_DISABLE_API`
| example | ```
# Command line example
sensu-agent start --disable-api

# /etc/sensu/agent.yml 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`
| example | ```
# Command line example
sensu-agent start --events-burst-limit 20

# /etc/sensu/agent.yml 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`

example	<pre># Command line example
	sensu-agent start --events-rate-limit 20.0
	
	# /etc/sensu/agent.yml example
	events-rate-limit: 20.0</pre>

Ephemeral agent configuration flags

deregister	
description	`true` if a deregistration event should be created upon Sensu agent process stop. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_DEREGISTER`

example	<pre># Command line example
	sensu-agent start --deregister
	
	# /etc/sensu/agent.yml example
	deregister: true</pre>

deregistration-handler

description	Name of a deregistration handler that processes agent deregistration events. This flag overrides any handlers applied by the `deregistration-handler` backend configuration flag.
type	String
environment variable	`SENSU_DEREGISTRATION_HANDLER`
| example | ```
# Command line example
sensu-agent start --deregistration-handler deregister

# /etc/sensu/agent.yml example
deregistration-handler: "deregister"
``` |

Keepalive configuration flags

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` .
type	Integer
default	`0`
environment variable	`SENSU_KEEPALIVE_CRITICAL_TIMEOUT`
| example | ```
# Command line example
sensu-agent start --keepalive-critical-timeout 300

# /etc/sensu/agent.yml example
keepalive-critical-timeout: 300
``` |

keepalive-interval

description	Number of seconds between keepalive events.
type	Integer
default	`20`
environment variable	`SENSU_KEEPALIVE_INTERNAL`
example	```# Command line example
sensu-agent start --keepalive-interval 30

/etc/sensu/agent.yml example
keepalive-interval: 30``` |

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`.
type	Integer
default	`120`
environment variable	`SENSU_KEEPALIVE_WARNING_TIMEOUT`
example	```# Command line example
sensu-agent start --keepalive-warning-timeout 300

/etc/sensu/agent.yml example
keepalive-warning-timeout: 300``` |

Security configuration flags

namespace	
description	Agent namespace. **NOTE**: *Agents are represented in the backend as a class of entity. Entities can only belong to a <u>single namespace</u>.*
type	String
default	`default`
environment variable	`SENSU_NAMESPACE`
example	```# Command line example sensu-agent start --namespace ops # /etc/sensu/agent.yml example namespace: "ops"```

user	
description	<u>Sensu RBAC username</u> 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`
example	```# Command line example sensu-agent start --user agent-01 # /etc/sensu/agent.yml example user: "agent-01"```

password

description	Sensu RBAC password used by the agent.
type	String
default	`P@ssw0rd!`
environment variable	`SENSU_PASSWORD`
| example | ```
# Command line example
sensu-agent start --password secure-password

# /etc/sensu/agent.yml 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`
| example | ```
# Command line example
sensu-agent start --redact secret,ec2_access_key

# /etc/sensu/agent.yml example
redact:
  - secret
  - ec2_access_key
``` |

cert-file

description	Path to the agent certificate file used in mutual TLS authentication.
type	String
default	`""`
environment variable	`SENSU_CERT_FILE`
example	

```
# Command line example
sensu-agent start --cert-file /path/to/agent-1.pem

# /etc/sensu/agent.yml example
cert-file: "/path/to/agent-1.pem"
```

trusted-ca-file

description	SSL/TLS certificate authority.
type	String
default	`""`
environment variable	`SENSU_TRUSTED_CA_FILE`
example	

```
# Command line example
sensu-agent start --trusted-ca-file /path/to/trusted-
certificate-authorities.pem

# /etc/sensu/agent.yml example
trusted-ca-file: "/path/to/trusted-certificate-
authorities.pem"
```

key-file

description	Path to the agent key file used in mutual TLS authentication.
type	String
default	`""`
environment variable	`SENSU_KEY_FILE`
example	

```
# Command line example
sensu-agent start --key-file /path/to/agent-1-key.pem

# /etc/sensu/agent.yml example
key-file: "/path/to/agent-1-key.pem"
```

insecure-skip-tls-verify

description	Skip SSL verification. ***WARNING***: *This configuration flag is intended for use in development systems only. Do not use this flag in production.*
type	Boolean
default	`false`
environment variable	`SENSU_INSECURE_SKIP_TLS_VERIFY`
example	

```
# Command line example
sensu-agent start --insecure-skip-tls-verify

# /etc/sensu/agent.yml example
insecure-skip-tls-verify: true
```

Socket configuration flags

socket-host

description	Address to bind the Sensu agent socket to.
type	String
default	`127.0.0.1`
environment variable	`SENSU_SOCKET_HOST`
| example | ```
# Command line example
sensu-agent start --socket-host 0.0.0.0

# /etc/sensu/agent.yml example
socket-host: "0.0.0.0"
``` |

socket-port

description	Port the Sensu agent socket listens on.
type	Integer
default	`3030`
environment variable	`SENSU_SOCKET_PORT`
| example | ```
# Command line example
sensu-agent start --socket-port 4030

# /etc/sensu/agent.yml example
socket-port: 4030
``` |

disable-sockets

description	`true` to disable the agent TCP and UDP event sockets. Othewise, `false`.

type	Boolean
default	`false`
environment variable	`SENSU_DISABLE_SOCKETS`
example	

```
# Command line example
sensu-agent start --disable-sockets

# /etc/sensu/agent.yml example
disable-sockets: true
```

StatsD configuration flags

statsd-disable

description	`true` to disable the StatsD listener and metrics server. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_STATSD_DISABLE`
example	

```
# Command line example
sensu-agent start --statsd-disable

# /etc/sensu/agent.yml example
statsd-disable: true
```

statsd-event-handlers

description	List of event handlers for StatsD metrics.

type	List
environment variable	`SENSU_STATSD_EVENT_HANDLERS`
example	

```
# Command line examples
sensu-agent start --statsd-event-handlers influxdb,opentsdb
sensu-agent start --statsd-event-handlers influxdb --statsd-
event-handlers opentsdb

# /etc/sensu/agent.yml 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`
example	

```
# Command line example
sensu-agent start --statsd-flush-interval 30

# /etc/sensu/agent.yml 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`
example	 ``` # Command line example sensu-agent start --statsd-metrics-host 0.0.0.0 # /etc/sensu/agent.yml example statsd-metrics-host: "0.0.0.0" ```

statsd-metrics-port

description	Port used for the StatsD metrics server.
type	Integer
default	`8125`
environment variable	`SENSU_STATSD_METRICS_PORT`
example	 ``` # Command line example sensu-agent start --statsd-metrics-port 6125 # /etc/sensu/agent.yml example statsd-metrics-port: 6125 ```

Allow list configuration

exec

description	Command to allow the Sensu agent to run as a check or a hook.
required	true

type	String
environment variable	`SENSU_EXEC`
example	`"exec": "/usr/local/bin/check_memory.sh"`

sha512

description	Checksum of the check or hook executable.
required	false
type	String
environment variable	`SENSU_SHA512`
example	`"sha512": "4f926bf4328..."`

args

description	Arguments for the `exec` command.
required	true
type	Array
environment variable	`SENSU_ARGS`
example	`"args": ["foo"]`

enable_env

description	`true` to enable environment variables. Otherwise, `false`.

required	false
type	Boolean
environment variable	`SENSU_ENABLE_ENV`
example	`"enable_env": true`

Example allow list configuration file

YML

```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

```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"
  }
]
```

Configuration via environment variables

The `sensu-agent` service configured by our supported packages will read environment variables
from `/etc/default/sensu-agent` on Debian/Ubuntu systems and `/etc/sysconfig/sensu-agent`
on RHEL systems. The installation package does not create these files, so you will need to create them.

UBUNTU/DEBIAN

```
$ sudo touch /etc/default/sensu-agent
```

RHEL/CENTOS

```
$ sudo touch /etc/sysconfig/sensu-agent
```

For any configuration flag you wish to specify as an environment variable, you must prepend `SENSU_`, convert dashes (`-`) to underscores (`_`), and capitalize all letters. Then, add the resulting environment variable to the appropriate environment file described above. You must restart the service for these settings to take effect.

In this example, the `api-host` flag is configured as an environment variable and set to `"0.0.0.0"`:

UBUNTU/DEBIAN

```
$ echo 'SENSU_API_HOST="0.0.0.0' | sudo tee -a /etc/default/sensu-agent
$ sudo systemctl restart sensu-agent
```

RHEL/CENTOS

```
$ echo 'SENSU_API_HOST="0.0.0.0' | sudo tee -a /etc/sysconfig/sensu-agent
$ sudo systemctl restart sensu-agent
```

Use environment variables with the Sensu agent

After you configure your sensu-agent service to read environment variables from `/etc/default/sensu-agent` (Debian/Ubuntu) or `/etc/sysconfig/sensu-agent` (RHEL), any environment variables you provide there will be available to check and hook commands executed by the Sensu agent. This includes your checks and plugins.

For example, if you configure a `SENSU_TEST_VAR` variable in your sensu-agent file, it will be available to use in your check configurations as `$SENSU_TEST_VAR`.

Sensu backend

The Sensu backend is a service that manages check requests and event data. Every Sensu backend includes an integrated transport for scheduling checks using subscriptions, an event processing pipeline that applies filters, mutators, and handlers, an embedded etcd datastore for storing configuration and state, a Sensu API, a Sensu dashboard, and the `sensu-backend` command line tool. The Sensu backend is available for Ubuntu/Debian and RHEL/CentOS distributions of Linux. See the installation guide to install the backend.

Create event pipelines

The backend processes event data and executes filters, mutators, and handlers. These pipelines are powerful tools to automate your monitoring workflows. To learn more about filters, mutators, and handlers, see:

- Guide to sending Slack alerts with handlers

- Guide to reducing alerting fatigue with filters

- Filters reference documentation

- Mutators reference documentation

- Handlers reference documentation

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, see:

- ¬ Monitor server resources with checks
- ¬ Collect metrics with checks
- ¬ Checks reference documentation

Initialization

For a **new** installation, you must set up an administrator username and password.To do this, set environment variables as shown below, replacing `YOUR_USERNAME` and `YOUR_PASSWORD` with the username and password you want to use:

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=YOUR_USERNAME
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=YOUR_PASSWORD
sensu-backend init
```

NOTE: *Make sure the Sensu backend is running before you run* `sensu-backend init`.

You can also run the `sensu-backend init` command in interactive mode if you prefer to respond to prompts for your username and password:

```
sensu-backend init --interactive

Admin Username: YOUR_USERNAME
Admin Password: YOUR_PASSWORD
```

This initialization step bootstraps the first admin user account for your Sensu installation.This account will be granted the cluster admin role.

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.*

To see available initialization flags:

```
sensu-backend init --help
```

Operation and service management

NOTE: *Commands in this section may require administrative privileges.*

Start the service

Use the `sensu-backend` tool to start the backend and apply configuration flags.

To start the backend with configuration flags:

```
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

To see available configuration flags and defaults:

```
sensu-backend start --help
```

If you do not provide any configuration flags, the backend loads configuration from `/etc/sensu/backend.yml` by default.

To start the backend using a service manager:

```
service sensu-backend start
```

Stop the service

To stop the backend service using a service manager:

```
service sensu-backend stop
```

Restart the service

You must restart the backend to implement any configuration updates.

To restart the backend using a service manager:

```
service sensu-backend restart
```

Enable on boot

To enable the backend to start on system boot:

```
systemctl enable sensu-backend
```

To disable the backend from starting on system boot:

```
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

To see the status of the backend service using a service manager:

```
service sensu-backend status
```

Get service version

To 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:

```
# Show sensu-backend commands
sensu-backend help

# Show options for the sensu-backend start subcommand
sensu-backend start --help
```

Cluster

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, see:

- ¬ Datastore configuration flags

- ¬ Run a Sensu cluster

Synchronize time

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.

Configuration

You can specify the backend configuration with either a `/etc/sensu/backend.yml` file or `sensu-backend start` configuration flags. The backend requires that the `state-dir` flag is set before starting. All other required flags have default values. See the example backend configuration file for flags and defaults. The backend loads configuration upon startup, so you must restart the backend for any configuration updates to take effect.

Configuration summary

```
$ sensu-backend start --help
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-host string                 agent listener host (default "[::]")
      --agent-port int                    agent listener port (default 8081)
      --agent-write-timeout int           timeout in seconds for agent writes
(default 15)
      --api-listen-address string         address to listen on for API traffic
(default "[::]:8080")
      --api-url string                    URL of the API to connect to (default
"http://localhost:8080")
      --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
          --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)
          --debug                               enable debugging and profiling features
          --deregistration-handler string     default deregistration handler
          --event-log-buffer-size int          buffer size of the event logger (default
100000)
          --event-log-file string               path to the event log file
          --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 JSON Web Tokens (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
          --log-level string                    logging level [panic, fatal, error,
warn, info, debug] (default "warn")
          --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)
      -d, --state-dir string                        path to sensu state storage (default
"/var/lib/sensu/sensu-backend")
          --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 the rest of the cluster (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-urls string                 client URLs to use when operating
as an etcd client
     --etcd-client-cert-auth                   enable client cert authentication
     --etcd-discovery                          use the dynamic cluster
configuration method etcd
discovery instead of the static `--initial-cluster method`
     --etcd-discovery-srv                      use the dynamic cluster
configuration method DNS SRV
discovery instead of the static `--initial-cluster method`
     --etcd-election-timeout uint              time in ms a follower node will
go without hearing a heartbeat before attempting to become leader itself (default
1000)
     --etcd-heartbeat-interval uint            interval in ms with which the
etcd leader will notify followers that it is still the leader (default 100)
     --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 (default "default=http://127.0.0.1:2380")
     --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-listen-client-urls strings         list of URLs to listen on for
client traffic (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-max-request-bytes uint             maximum etcd request size in
bytes (use with caution; default 1572864)
     --etcd-name string                        human-readable name for this
member (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
      --no-embed-etcd                            don't embed etcd; use external
etcd instead
```

General configuration flags

api-listen-address	
description	Address the API daemon will listen for requests on.
type	String
default	`[::]:8080`
environment variable	`SENSU_BACKEND_API_LISTEN_ADDRESS`
example	```# Command line example
sensu-backend start --api-listen-address [::]:8080

/etc/sensu/backend.yml example
api-listen-address: "[::]:8080"``` |

api-url	
description	URL used to connect to the API.
type	String
default	`http://localhost:8080`
environment variable	`SENSU_BACKEND_API_URL`
example	```# Command line example
sensu-backend start --api-url http://localhost:8080```

```
# /etc/sensu/backend.yml example
api-url: "http://localhost:8080"
```

cache-dir

description	Path to store cached data.
type	String
default	`/var/cache/sensu/sensu-backend`
environment variable	`SENSU_BACKEND_CACHE_DIR`
example	

```
# Command line example
sensu-backend start --cache-dir /cache/sensu-backend

# /etc/sensu/backend.yml example
cache-dir: "/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`
example	

```
# Command line example
sensu-backend start --config-file /etc/sensu/backend.yml
sensu-backend start -c /etc/sensu/backend.yml

# /etc/sensu/backend.yml example
config-file: "/etc/sensu/backend.yml"
```

debug

description	If `true`, enable debugging and profiling features. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_DEBUG`
example	

```
# Command line example
sensu-backend start --debug


# /etc/sensu/backend.yml example
debug: true
```

deregistration-handler

description	Default event handler to use when processing agent deregistration events.
type	String
default	`""`
environment variable	`SENSU_BACKEND_DEREGISTRATION_HANDLER`
example	

```
# Command line example
sensu-backend start --deregistration-handler
/path/to/handler.sh


# /etc/sensu/backend.yml example
deregistration-handler: "/path/to/handler.sh"
```

log-level

description	Logging level: `panic`, `fatal`, `error`, `warn`, `info`, or `debug`.
type	String
default	`warn`
environment variable	`SENSU_BACKEND_LOG_LEVEL`
example	

```
# Command line example
sensu-backend start --log-level debug

# /etc/sensu/backend.yml example
log-level: "debug"
```

state-dir

description	Path to Sensu state storage: `/var/lib/sensu/sensu-backend`.
type	String
required	true
environment variable	`SENSU_BACKEND_STATE_DIR`
example	

```
# Command line example
sensu-backend start --state-dir /var/lib/sensu/sensu-backend
sensu-backend start -d /var/lib/sensu/sensu-backend

# /etc/sensu/backend.yml example
state-dir: "/var/lib/sensu/sensu-backend"
```

Agent communication configuration flags

agent-auth-cert-file

description	TLS certificate in PEM format for agent certificate authentication.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_CERT_FILE`
example	```# Command line example
sensu-backend start --agent-auth-cert-file
/path/to/ssl/cert.pem

/etc/sensu/backend.yml example
agent-auth-cert-file: /path/to/ssl/cert.pem``` |

agent-auth-crl-urls

description	URLs of CRLs for agent certificate authentication.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_CRL_URLS`
example	```# Command line example
sensu-backend start --agent-auth-crl-urls
http://localhost/CARoot.crl

/etc/sensu/backend.yml 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`

example	<pre># Command line example
	sensu-backend start --agent-auth-key-file
	/path/to/ssl/key.pem
	
	# /etc/sensu/backend.yml example
	agent-auth-key-file: /path/to/ssl/key.pem</pre>

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`

example	<pre># Command line example
	sensu-backend start --agent-auth-trusted-ca-file
	/path/to/ssl/ca.pem
	
	# /etc/sensu/backend.yml example
	agent-auth-trusted-ca-file: /path/to/ssl/ca.pem</pre>

agent-host

description	Agent listener host. Listens on all IPv4 and IPv6 addresses by default.
type	String
default	`[::]`
environment variable	`SENSU_BACKEND_AGENT_HOST`
| example | ```
# Command line example
sensu-backend start --agent-host 127.0.0.1

# /etc/sensu/backend.yml example
agent-host: "127.0.0.1"
``` |

agent-port

description	Agent listener port.
type	Integer
default	`8081`
environment variable	`SENSU_BACKEND_AGENT_PORT`
| example | ```
# Command line example
sensu-backend start --agent-port 8081

# /etc/sensu/backend.yml example
agent-port: 8081
``` |

Security configuration flags

cert-file

description	Path to the primary backend certificate file. Specifies a fallback SSL/TLS certificate if the flag `dashboard-cert-file` is not used. This certificate

	secures communications between the Sensu dashboard and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`
environment variable	`SENSU_BACKEND_CERT_FILE`
| example | ```
# Command line example
sensu-backend start --cert-file /path/to/ssl/cert.pem

# /etc/sensu/backend.yml example
cert-file: "/path/to/ssl/cert.pem"
``` |

insecure-skip-tls-verify

description	If `true`, skip SSL verification. Otherwise, `false`. ***WARNING**: This configuration flag is intended for use in development systems only. Do not use this flag in production.*
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_INSECURE_SKIP_TLS_VERIFY`
| example | ```
# Command line example
sensu-backend start --insecure-skip-tls-verify

# /etc/sensu/backend.yml 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`
| example | ```
# Command line example
sensu-backend start --jwt-private-key-file
/path/to/key/private.pem

# /etc/sensu/backend.yml 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
| example | ```
# Command line example
sensu-backend start --jwt-public-key-file
/path/to/key/public.pem

# /etc/sensu/backend.yml 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 flag `dashboard-key-file` is not used. This key secures communication between the Sensu dashboard and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`
environment variable	`SENSU_BACKEND_KEY_FILE`
| example | ```
# Command line example
sensu-backend start --key-file /path/to/ssl/key.pem

# /etc/sensu/backend.yml example
key-file: "/path/to/ssl/key.pem"
``` |

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 dashboard 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`
| example | ```
# Command line example
sensu-backend start --trusted-ca-file /path/to/trusted-
certificate-authorities.pem
``` |

```
# /etc/sensu/backend.yml example
trusted-ca-file: "/path/to/trusted-certificate-
authorities.pem"
```

Dashboard configuration flags

dashboard-cert-file	
description	Dashboard TLS certificate in PEM format. This certificate secures communication with the Sensu dashboard. If the `dashboard-cert-file` is not provided in the backend configuration, Sensu uses the certificate specified in the `cert-file` flag for the dashboard.
type	String
default	`""`
environment variable	`SENSU_BACKEND_DASHBOARD_CERT_FILE`

example	<pre># Command line example
	sensu-backend start --dashboard-cert-file
	/path/to/tls/cert.pem
	
	# /etc/sensu/backend.yml example
	dashboard-cert-file: "/path/to/tls/cert.pem"</pre>

dashboard-host	
description	Dashboard listener host.
type	String
default	`[::]`
environment variable	`SENSU_BACKEND_DASHBOARD_HOST`

example	
| | ```
# Command line example
sensu-backend start --dashboard-host 127.0.0.1

# /etc/sensu/backend.yml example
dashboard-host: "127.0.0.1"
``` |

dashboard-key-file

description	Dashboard TLS certificate key in PEM format. This key secures communication with the Sensu dashboard. If the `dashboard-key-file` is not provided in the backend configuration, Sensu uses the key specified in the `key-file` flag for the dashboard.
type	String
default	`""`
environment variable	`SENSU_BACKEND_DASHBOARD_KEY_FILE`
| example | ```
# Command line example
sensu-backend start --dashboard-key-file
/path/to/tls/key.pem

# /etc/sensu/backend.yml example
dashboard-key-file: "/path/to/tls/key.pem"
``` |

dashboard-port

description	Dashboard listener port.
type	Integer
default	`3000`
environment variable	`SENSU_BACKEND_DASHBOARD_PORT`
example	

```
# Command line example
sensu-backend start --dashboard-port 4000


# /etc/sensu/backend.yml example
dashboard-port: 4000
```

Datastore and cluster configuration flags

etcd-advertise-client-urls	
description	List of this member's client URLs to advertise to the rest of the cluster.
type	List
default	`http://localhost:2379`
environment variable	`SENSU_BACKEND_ETCD_ADVERTISE_CLIENT_URLS`
example	```# Command line examples
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

/etc/sensu/backend.yml 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.

type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_CERT_FILE`
| example | ```
# Command line example
sensu-backend start --etcd-cert-file ./client.pem

# /etc/sensu/backend.yml example
etcd-cert-file: "./client.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"` .
| 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` |
| example | ```
# Command line examples
sensu-backend start --etcd-cipher-suites
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AE
S_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

# /etc/sensu/backend.yml 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`.
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_ETCD_CLIENT_CERT_AUTH`

example	<pre># Command line example
	sensu-backend start --etcd-client-cert-auth
	
	# /etc/sensu/backend.yml example
	etcd-client-cert-auth: true</pre>

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 flag in conjunction with `--no-embed-etcd` when executing sensu-backend start or sensu-backend init. If you do not use this flag when using `--no-embed-etcd`, sensu-backend start and sensu-backend-init will fall back to –etcd-listen-client-urls.
type	List

default	`http://127.0.0.1:2379`
environment variable	`SENSU_BACKEND_ETCD_CLIENT_URLS`
| example | ```
# Command line examples
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

# /etc/sensu/backend.yml 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.
type	String
default	""
environment variable	`SENSU_BACKEND_ETCD_DISCOVERY`
| example | ```
# Command line example
sensu-backend start --etcd-discovery
https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de

# /etc/sensu/backend.yml 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.
type	String
default	""
environment variable	`SENSU_BACKEND_ETCD_DISCOVERY_SRV`
example	

```
# Command line example
sensu-backend start --etcd-discovery-srv example.org

# /etc/sensu/backend.yml 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.
type	List
default	`http://127.0.0.1:2380`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_ADVERTISE_PEER_URLS`
example	

```
# Command line examples
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

# /etc/sensu/backend.yml 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.
type	String
default	`default=http://127.0.0.1:2380`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER`
example	```# 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

/etc/sensu/backend.yml 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`).
type	String
default	`new`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER_STATE`
example	```# Command line example
sensu-backend start --etcd-initial-cluster-state existing```

```
# /etc/sensu/backend.yml example
etcd-initial-cluster-state: "existing"
```

etcd-initial-cluster-token

description	Initial cluster token for the etcd cluster during bootstrap.
type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER_TOKEN`

example	<pre># Command line example
	sensu-backend start --etcd-initial-cluster-token sensu
	
	# /etc/sensu/backend.yml example
	etcd-initial-cluster-token: "sensu"</pre>

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.
type	String
environment variable	`SENSU_BACKEND_ETCD_KEY_FILE`

example	<pre># Command line example
	sensu-backend start --etcd-key-file ./client-key.pem
	
	# /etc/sensu/backend.yml example
	etcd-key-file: "./client-key.pem"</pre>
```

## etcd-listen-client-urls

| | |
|---|---|
| description | List of URLs to listen on for client traffic. |
| type | List |
| default | `http://127.0.0.1:2379` |
| environment variable | `SENSU_BACKEND_ETCD_LISTEN_CLIENT_URLS` |
| example | |

```
# Command line examples
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

# /etc/sensu/backend.yml 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.
type	List
default	`http://127.0.0.1:2380`
environment variable	`SENSU_BACKEND_ETCD_LISTEN_PEER_URLS`
example	

```
# Command line examples
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

# /etc/sensu/backend.yml example
etcd-listen-peer-urls:
  - https://10.0.0.1:2380
  - https://10.1.0.1:2380
```

etcd-name

description	Human-readable name for this member.
type	String
default	`default`
environment variable	`SENSU_BACKEND_ETCD_NAME`
example	```# Command line example
sensu-backend start --etcd-name backend-0

/etc/sensu/backend.yml example
etcd-name: "backend-0"``` |

etcd-peer-cert-file

description	Path to the peer server TLS certificate file.
type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_CERT_FILE`
example	```# Command line example
sensu-backend start --etcd-peer-cert-file ./backend-0.pem```

```
# /etc/sensu/backend.yml example
etcd-peer-cert-file: "./backend-0.pem"
```

etcd-peer-client-cert-auth

description	Enable peer client certificate authentication.
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_ETCD_PEER_CLIENT_CERT_AUTH`
example	```# Command line example
sensu-backend start --etcd-peer-client-cert-auth

/etc/sensu/backend.yml 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.
type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_KEY_FILE`
example	```# Command line example
sensu-backend start --etcd-peer-key-file ./backend-0-key.pem

/etc/sensu/backend.yml example
etcd-peer-key-file: "./backend-0-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.
type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_TRUSTED_CA_FILE`
example	```# Command line example
sensu-backend start --etcd-peer-trusted-ca-file ./ca.pem

/etc/sensu/backend.yml 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.
type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_TRUSTED_CA_FILE`
example	```# Command line example
sensu-backend start --etcd-trusted-ca-file ./ca.pem

/etc/sensu/backend.yml example
etcd-trusted-ca-file: "./ca.pem"``` |

no-embed-etcd

description	If `true`, do not embed etcd (use external etcd instead). Otherwise,

`false` .

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_NO_EMBED_ETCD`

example	<pre># Command line example
	sensu-backend start --no-embed-etcd
	
	# /etc/sensu/backend.yml example
	no-embed-etcd: true</pre>

Advanced configuration options

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`

example	<pre># Command line example
	sensu-backend start --eventd-buffer-size 100
	
	# /etc/sensu/backend.yml example
	eventd-buffer-size: 100</pre>

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`
example	```# Command line example
sensu-backend start --eventd-workers 100

/etc/sensu/backend.yml 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`
example	```# Command line example
sensu-backend start --keepalived-buffer-size 100

/etc/sensu/backend.yml 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`
| example | ```
# Command line example
sensu-backend start --keepalived-workers 100

# /etc/sensu/backend.yml 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`
| example | ```
# Command line example
sensu-backend start --pipelined-buffer-size 100

# /etc/sensu/backend.yml 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`
example	```# Command line example\nsensu-backend start --pipelined-workers 100\n\n# /etc/sensu/backend.yml example\npipelined-workers: 100```

etcd-election-timeout

description	Time that a follower node will go without hearing a heartbeat before attempting to become leader itself. In milliseconds (ms). See 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.*
type	Integer
default	`1000`
environment variable	`SENSU_BACKEND_ETCD_ELECTION_TIMEOUT`
example	```# Command line example\nsensu-backend start --etcd-election-timeout 1000\n\n# /etc/sensu/backend.yml example```

```
etcd-election-timeout: 1000
```

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. See 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.*
type	Integer
default	`100`
environment variable	`SENSU_BACKEND_ETCD_HEARTBEAT_INTERVAL`
example	```# Command line example
sensu-backend start --etcd-heartbeat-interval 100

/etc/sensu/backend.yml example
etcd-heartbeat-interval: 100``` |

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.*
type	Integer
default	`1572864`

environment variable	`SENSU_BACKEND_ETCD_MAX_REQUEST_BYTES`
| example | ```
# Command line example
sensu-backend start --etcd-max-request-bytes 1572864

# /etc/sensu/backend.yml 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.
type	Integer
default	`4294967296`
environment variable	`SENSU_BACKEND_ETCD_QUOTA_BACKEND_BYTES`
| example | ```
# Command line example
sensu-backend start --etcd-quota-backend-bytes 4294967296

# /etc/sensu/backend.yml example
etcd-quota-backend-bytes: 4294967296
``` |

Configuration via environment variables

The `sensu-backend` service configured by our supported packages will read environment variables from `/etc/default/sensu-backend` on Debian/Ubuntu systems and `/etc/sysconfig/sensu-backend` on RHEL systems. The installation package does not create these files, so you will need to create them.

UBUNTU/DEBIAN

```
$ sudo touch /etc/default/sensu-backend
```

RHEL/CENTOS

```
$ sudo touch /etc/sysconfig/sensu-backend
```

For any backend configuration flag you wish to specify as an environment variable, you must prepend `SENSU_BACKEND_`, convert dashes (`-`) to underscores (`_`), and capitalize all letters. Then, add the resulting environment variable to the appropriate environment file described above. You must restart the service for these settings to take effect.

In this example, the `api-listen-address` flag is configured as an environment variable and set to `192.168.100.20:8080`:

UBUNTU/DEBIAN

```
$ echo 'SENSU_BACKEND_API_LISTEN_ADDRESS=192.168.100.20:8080' | sudo tee -a
/etc/default/sensu-backend
$ sudo systemctl restart sensu-backend
```

RHEL/CENTOS

```
$ echo 'SENSU_BACKEND_API_LISTEN_ADDRESS=192.168.100.20:8080' | sudo tee -a
/etc/sysconfig/sensu-backend
$ sudo systemctl restart sensu-backend
```

Event logging

COMMERCIAL FEATURE: Access event logging in the packaged Sensu Go distribution. For more information, see Get started with commercial features.

If you wish, you can log all Sensu events to a file in JSON format. You can use this file as an input source for your favorite data lake solution. The event logging functionality provides better performance and reliability than event handlers.

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`
example	

```
# Command line example
sensu-backend start --event-log-buffer-size 100000


# /etc/sensu/backend.yml example
event-log-buffer-size: 100000
```

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`
example	

```
# Command line example
sensu-backend start --event-log-file
/var/log/sensu/events.log


# /etc/sensu/backend.yml example
event-log-file: "/var/log/sensu/events.log"
```

Log rotation

Event logging supports log rotation via the *SIGHUP* signal. 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.

Here are some log rotate sample configurations:

systemd

```
/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
}
```

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
}
```

API Keys

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.

Use the APIKey API to create, retrieve, and delete API keys.

Authorization header format

Use the following header format to authenticate with API keys, replacing `API_KEY` with your API key value:

```
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 resource structure
YML

```
type: APIKey
api_version: core/v2
metadata:
  name: 19803eb8-36a6-4203-a225-28ec4e9f4444
spec:
  created_at: 1570732266
  username: admin
```

JSON

```json
{
  "type": "APIKey",
  "api_version": "core/v2",
  "metadata" : {
    "name": "19803eb8-36a6-4203-a225-28ec4e9f4444"
  },
  "spec": {
    "created_at": 1570732266,
    "username": "admin"
  }
}
```

API key specification

Top-level attributes

type	
description	Top-level attribute that specifies the resource type. API keys should always be type `APIKey` .
required	true
type	String
example	`"type": "APIKey"`

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	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the API key, including the `name`. 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	`"metadata": {`
	`  "name": "19803eb8-36a6-4203-a225-28ec4e9f4444"`
	`}`

spec

description	Top-level map that includes the API key's spec attributes.
required	true
type	Map of key-value pairs
example	

```
"spec": {
    "created_at": 1570732266,
    "username": "admin"
  }
```

Metadata attributes

name	
description	Unique string used to identify the API key. Sensu randomly generates a UUID for the `name` value — users cannot provide a name for an API key.
required	true
type	String
example	`"name": "19803eb8-36a6-4203-a225-28ec4e9f4444"`

Spec attributes

username	
description	User associated with the API key.
required	true
type	Array
example	`"username": "admin"`

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	`"created_at": 1234567890`

Examples

YML

```yaml
type: APIKey
api_version: core/v2
metadata:
  name: 19803eb8-36a6-4203-a225-28ec4e9f4444
spec:
  created_at: 1570732266
  username: admin
```

JSON

```json
{
  "type": "APIKey",
  "api_version": "core/v2",
  "metadata": {
    "name": "19803eb8-36a6-4203-a225-28ec4e9f4444"
  },
  "spec": {
    "created_at": 1570732266,
    "username": "admin"
  }
}
```

Assets

You can discover, download, and share assets using Bonsai, the Sensu asset index.Read Install plugins with assets to get started.

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. Sensu supports runtime assets for checks, filters, mutators, and handlers.

NOTE: Assets are not required to use Sensu Go in production. You can install Sensu plugins using the sensu-install tool or a configuration management solution.

The Sensu backend executes handler, filter, and mutator assets.The Sensu agent executes check assets.At runtime, the backend or agent sequentially evaluates assets that appear in the `runtime_assets` attribute of the handler, filter, mutator, or check being executed.

Asset builds

An 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: 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 build evaluation

For each build provided in an 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 assets with multiple builds, only the first build which matches will be downloaded and installed.

Asset build installation

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` flag. Disable assets for an agent with the agent `--disable-assets` configuration flag.

Asset build execution

The directory path of each asset defined in `runtime_assets` is appended to the `PATH` before the handler, filter, mutator, or check `command` is executed. Subsequent handler, filter, mutator, or check executions look for the asset in the local cache and ensure that the contents match the configured checksum.

See the example asset with a check for a use case with a Sensu resource (a check) and an asset.

Asset format specification

Sensu expects an 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. See the Sensu Go Plugin template for an example 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 an 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 an asset, see 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 asset is not in the local cache, it is downloaded from the asset URL. The Sensu backend does not currently provide any storage for assets. Sensu expects assets to be retrieved over HTTP or HTTPS.

Example asset structure

```
sensu-example-handler_1.0.0_linux_amd64
├── CHANGELOG.md
├── LICENSE
├── README.md
└── bin
  └── my-check.sh
└── lib
└── include
```

Asset hello world example

In this example, you'll run a script that outputs `Hello World`:

```
hello-world.sh
```

```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 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. Create the directories `sensu-go-hello-world` and `/bin` :

```
$ mkdir sensu-go-hello-world

$ cd sensu-go-hello-world

$ mkdir bin

$ cp hello-world.sh bin/

$ tree
.
└── bin
    └── hello-world.sh
```

Next, make sure that the script is marked as executable:

```
$ chmod +x bin/hello-world.sh
mode of 'hello-world.sh' changed from 0644 (rw-r--r--) to 0755 (rwxr-xr-x)
```

Now that the script is in the directory, move on to the next step: packaging the `sensu-go-hello-world` directory as an asset tarball.

Package the asset

Assets are archives, so the first step in packaging the asset is to create a tar.gz archive of your project. This assumes you're in the directory you want to tar up:

```
$ cd ..
$ tar -C sensu-go-hello-world -cvzf sensu-go-hello-world-0.0.1.tar.gz .
...
```

Now that you've created an archive, you need to generate a SHA512 sum for it (this is required for the asset to work):

```
sha512sum sensu-go-hello-world-0.0.1.tar.gz | tee sha512sum.txt
dbfd4a714c0c51c57f77daeb62f4a21141665ae71440951399be2d899bf44b3634dad2e6f2516fff1ef4
b154c198b9c7cdfe1e8867788c820db7bb5bcad83827 sensu-go-hello-world-0.0.1.tar.gz
```

From here, you can host your 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 on Discourse.

To host your 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 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. 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	``` "type": "Asset" ```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For 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
example	``` "api_version": "core/v2" ```

metadata

description	Top-level collection of metadata about the asset, including the `name` and `namespace` 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. See 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

example	```
	"metadata": {
	  "name": "check_script",
	  "namespace": "default",
	``

```
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "playbook" : "www.example.url"
    }
  }
```

spec

description	Top-level map that includes the 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)	

```
"spec": {
  "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"
        ]
      }
    ],
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
```

| example (single build, deprecated) | ```
"spec": {
  "url": "http://example.com/asset.tar.gz",
  "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a
3ef3c2e8d154812246e5dda4a87450576b2c58ad9ab40c9e2edc31b288d
066b195b21b",
  "filters": [
    "entity.system.os == 'linux'",
    "entity.system.arch == 'amd64'"
  ],
  "headers": {
    "Authorization": "Bearer $TOKEN",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  }
}
``` |

Metadata attributes

name	
description	Unique name of the asset, validated with Go regex `\A[\w\.\-]+\z`.
required	true
type	String

example	
	`"name": "check_script"`

namespace

description	Sensu RBAC namespace that the asset belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering and tokens. If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	`"labels": {`

```
    "environment": "development",
    "region": "us-west-2"
  }
```

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

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

Spec attributes

builds

description	List of 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	`"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 asset.
required	true, unless `builds` are provided
type	String
example	

```
"url": "http://example.com/asset.tar.gz"
```

sha512

description	Checksum of the asset.
required	true, unless `builds` are provided
type	String
example	`"sha512": "4f926bf4328..."`

filters

description	Set of Sensu query expressions used to determine if the asset should be installed. If multiple expressions are included, each expression must return `true` for Sensu to install the asset. Filters for *check* assets should match agent entity platforms. Filters for *handler* and *filter* 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**: *Asset filters let you reuse checks across platforms safely. Assign 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	`"filters": ["entity.system.os=='linux'",` `"entity.system.arch=='amd64'"]`

headers

description	HTTP headers to apply to asset retrieval requests. You can use headers to access secured assets. For headers that require multiple values, separate the values with a comma.

required	false
type	Map of key-value string pairs
| example | ```
"headers": {
  "Authorization": "Bearer $TOKEN",
  "X-Forwarded-For": "client1, proxy1, proxy2"
}
``` |

Examples

Minimum required asset attributes

YML

```yaml
type: Asset
api_version: core/v2
metadata:
  name: check_script
  namespace: default
spec:
  builds:
  - sha512:
4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4a8
7450576b2c58ad9ab40c9e2edc31b288d066b195b21b
    url: http://example.com/asset.tar.gz
```

JSON

```json
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_script",
    "namespace": "default"
  },
  "spec": {
    "builds": [
```

```
    {
      "url": "http://example.com/asset.tar.gz",
      "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4a
87450576b2c58ad9ab40c9e2edc31b288d066b195b21b"
    }
  ]
  }
}
```

Asset definition (single-build, deprecated)

YML

```yaml
type: Asset
api_version: core/v2
metadata:
  name: check_cpu_linux_amd64
  namespace: default
  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 $TOKEN
    X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```
{
```

```json
    "type": "Asset",
    "api_version": "core/v2",
    "metadata": {
      "name": "check_cpu_linux_amd64",
      "namespace": "default",
      "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 $TOKEN",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    }
}
```

Asset definition (multiple-builds)
YML

```yaml
type: Asset
api_version: core/v2
metadata:
  name: check_cpu
  namespace: default
  labels:
    origin: bonsai
```

```yaml
  annotations:
    project_url: https://bonsai.sensu.io/assets/asachs01/sensu-go-cpu-check
    version: 0.0.3
spec:
  builds:
  - 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 $TOKEN
      X-Forwarded-For: client1, proxy1, proxy2
  - url:
https://assets.bonsai.sensu.io/981307deb10ebf1f1433a80da5504c3c53d5c44f/sensu-go-
cpu-check_0.0.3_linux_armv7.tar.gz
    sha512:
70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d1538374066a4603fc5566737bfd6c7ffa18
314edb858a9f93330a57d430deeb7fd6f75670a8c68b
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'arm'
    - entity.system.arm_version == 7
    headers:
      Authorization: Bearer $TOKEN
      X-Forwarded-For: client1, proxy1, proxy2
  - url:
https://assets.bonsai.sensu.io/981307deb10ebf1f1433a80da5504c3c53d5c44f/sensu-go-
cpu-check_0.0.3_windows_amd64.tar.gz
    sha512:
10d6411e5c8bd61349897cf8868087189e9ba59c3c206257e1ebc1300706539cf37524ac976d0ed9c809
9bdddc50efadacf4f3c89b04a1a8bf5db581f19c157f
    filters:
    - entity.system.os == 'windows'
    - entity.system.arch == 'amd64'
    headers:
      Authorization: Bearer $TOKEN
      X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu",
    "namespace": "default",
    "labels": {
      "origin": "bonsai"
    },
    "annotations": {
      "project_url": "https://bonsai.sensu.io/assets/asachs01/sensu-go-cpu-check",
      "version": "0.0.3"
    }
  },
  "spec": {
    "builds": [
      {
        "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 $TOKEN",
          "X-Forwarded-For": "client1, proxy1, proxy2"
        }
      },
      {
        "url":
"https://assets.bonsai.sensu.io/981307deb10ebf1f1433a80da5504c3c53d5c44f/sensu-go-
cpu-check_0.0.3_linux_armv7.tar.gz",
        "sha512":
"70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d1538374066a4603fc5566737bfd6c7ffa1
8314edb858a9f93330a57d430deeb7fd6f75670a8c68b",
```

```json
      "filters": [
        "entity.system.os == 'linux'",
        "entity.system.arch == 'arm'",
        "entity.system.arm_version == 7"
      ],
      "headers": {
        "Authorization": "Bearer $TOKEN",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    },
    {
      "url":
"https://assets.bonsai.sensu.io/981307deb10ebf1f1433a80da5504c3c53d5c44f/sensu-go-
cpu-check_0.0.3_windows_amd64.tar.gz",
      "sha512":
"10d6411e5c8bd61349897cf8868087189e9ba59c3c206257e1ebc1300706539cf37524ac976d0ed9c80
99bdddc50efadacf4f3c89b04a1a8bf5db581f19c157f",
      "filters": [
        "entity.system.os == 'windows'",
        "entity.system.arch == 'amd64'"
      ],
      "headers": {
        "Authorization": "Bearer $TOKEN",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    }
  ]
  }
}
```

Example asset with a check

YML

```yaml
---
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
  namespace: default
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
```

WRAPPED-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",
    "namespace": "default"
  },
  "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"
    ]
  }
}
```

Share an asset on Bonsai

Share your open-source assets on Bonsai and connect with the Sensu community.Bonsai supports

assets hosted on GitHub and released using GitHub releases. For more information about creating Sensu Plugins, see the Sensu Plugin specification.

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 asset template using Go and GoReleaser, see the Sensu Go plugin skeleton.

To share your asset on Bonsai, log in to Bonsai with your GitHub account and authorize Sensu. After you are logged in, you can register your asset on Bonsai by adding the GitHub repository, a description, and tags. Make sure to provide a helpful README for your 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 asset details per platform.
required	true
type	Array
example	(see below)

```yaml
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 asset.
required	true
type	String
example	`- platform: "linux"`
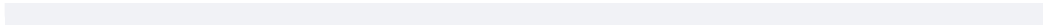
arch

description	Architecture supported by the asset.
required	true
type	String
example	

```
arch: "amd64"
```

asset_filename

description	File name of the archive that contains the asset.
required	true
type	String
example	

```
asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
```

sha_filename

description	SHA-512 checksum for the 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 assets

As of Sensu Go 5.12, you can delete assets with the `/assets (DELETE)` endpoint or via `sensuctl` (`sensuctl asset delete`).When you remove an asset from Sensu, this *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 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 asset from checks and hooks will surface in the event data. Errors as a result of failing to remove the asset reference on a mutator, handler, or filter will only surface in the backend logs.

Deleting an asset does not delete the archive or downloaded files on disk.You must remove the archive and downloaded files from the asset cache manually.

Checks

Checks work with Sensu agents to produce monitoring 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 index, to discover, download, and share Sensu check assets.

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 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 <u>STDOUT or STDERR</u>.
    - ⌐ 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 <u>publish-subscribe model</u>.

Subscriptions

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.Subscriptions typically correspond to a specific role or responsibility (for example. a webserver or database).

Subscriptions are powerful primitives in the monitoring context because they allow you to effectively monitor for specific behaviors or characteristics that correspond to the function provided by a particular system.For example, disk capacity thresholds might be more important (or at least different) on a database server than on a webserver.Conversely, CPU or memory usage thresholds might be more important on a caching system than on a file server.

Subscriptions also allow you to configure check requests for an entire group or subgroup of systems rather than requiring a traditional one-to-one mapping.

To configure subscriptions for a check, use the `subscriptions` attribute to specify an array of one or more subscription names.Sensu schedules checks once per interval for each agent with a matching subscription.For example, if we have three agents configured with the `system` subscription, a check configured with the `system` subscription results in three monitoring events per interval: one check execution per agent per interval.For Sensu to execute a check, the check definition must include a subscription that matches the subscription of at least one Sensu agent.

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 monitoring 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.*

Scheduling

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).

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

```yaml
type: CheckConfig
api_version: core/v2
metadata:
  name: interval_check
  namespace: default
spec:
  command: check-cpu.sh -w 75 -c 90
  handlers:
  - slack
  interval: 60
  publish: true
  subscriptions:
  - system
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "interval_check",
    "namespace": "default"
  },
  "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 (e.g.* `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

```yml
type: CheckConfig
api_version: core/v2
metadata:
  name: cron_check
  namespace: default
spec:
  command: check-cpu.sh -w 75 -c 90
  cron: '* * * * *'
  handlers:
  - slack
  publish: true
  subscriptions:
  - system
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "cron_check",
    "namespace": "default"
  },
```

```
    "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
  namespace: default
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
  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",
        "namespace": "default"
    },
    "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,
        "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 the checks API. 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

```yaml
type: CheckConfig
api_version: core/v2
metadata:
  name: ad_hoc_check
  namespace: default
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",
    "namespace": "default"
  },
  "spec": {
    "command": "check-cpu.sh -w 75 -c 90",
    "subscriptions": ["system"],
    "handlers": ["slack"],
    "interval": 60,
    "publish": false
  }
}
```

Proxy checks

Sensu supports running proxy checks where the results are considered to be for an entity that isn't

actually the one executing the check, regardless of whether that entity is a Sensu 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 the `proxy_entity_name` attribute or the `proxy_requests` attributes.

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 event is received by the backend.To avoid duplicate events, we recommend using the `round_robin` attribute with proxy checks.

Example proxy check using a `proxy_entity_name`

The following proxy check runs every 60 seconds, cycling through the agents with the `proxy` subscription alphabetically according to the agent name, for the proxy entity `sensu-site`.

YML

```yml
type: CheckConfig
api_version: core/v2
metadata:
  name: proxy_check
  namespace: default
spec:
  command: http_check.sh https://sensu.io
  handlers:
  - slack
  interval: 60
  proxy_entity_name: sensu-site
  publish: true
  round_robin: true
  subscriptions:
  - proxy
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
```

```
    "name": "proxy_check",
    "namespace": "default"
  },
  "spec": {
    "command": "http_check.sh https://sensu.io",
    "subscriptions": ["proxy"],
    "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 definitions specified in the `entity_attributes`, resulting in monitoring 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 still use Sensu query expressions to perform more complicated filtering on the available value, such as finding entities with particular subscriptions.

The `proxy_requests` attributes are a great way to monitor multiple entities using a single check definition when combined with token substitution.Because checks that include `proxy_requests` attributes need to be executed for each matching entity, we recommend using the `round_robin` attribute to distribute the check execution workload evenly across your Sensu agents.

Example proxy check using `proxy_requests`

The following proxy check runs every 60 seconds, cycling through the agents with the `proxy` 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.See the entity reference for information about using custom labels.

YML

```
type: CheckConfig
api_version: core/v2
metadata:
```

```yaml
  name: proxy_check_proxy_requests
  namespace: default
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:
  - proxy
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "proxy_check_proxy_requests",
    "namespace": "default"
  },
  "spec": {
    "command": "http_check.sh {{ .labels.url }}",
    "subscriptions": ["proxy"],
    "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

Sensu supports distributing proxy check executions across an interval using the `splay` and `splay_coverage` attributes. For example, if you assume that the `proxy_check_proxy_requests` check in the example above matches three proxy entities, you'd expect to see a burst of three events every 60 seconds. If you add the `splay` attribute (set to `true`) and the `splay_coverage` attribute (set to `90`) to the `proxy_requests` scope, Sensu will distribute the three check executions over 90% of the 60-second interval, resulting in three events splayed evenly across a 54-second period.

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.

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 (e.g. `0`, `1`, or `2`).

Learn how to use check hooks with the Sensu hooks reference documentation.

Check specification

Top-level attributes

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	`"type": "CheckConfig"`

api_version

description	Top-level attribute that specifies the Sensu API group and version. For checks in Sensu backend version 5.4 and later, 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	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the check, including the `name` and `namespace` 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. See 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	

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

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	```"spec": { "command": "/etc/sensu/plugins/check-chef-client.go", "interval": 10, "publish": true, "subscriptions": ["production"] }```

Metadata attributes

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	`"name": "check-cpu"`

namespace

description	Sensu RBAC namespace that the check belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering and tokens.
	If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	
| | ```
"labels": {
  "environment": "development",
  "region": "us-west-2"
}
``` |

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

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

Spec attributes

command

description	Check command to be executed.
required	true

type	String
example	`"command": "/etc/sensu/plugins/check-chef-client.go"`

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
example	`"subscriptions": ["production"]`

handlers

description	Array of Sensu event handlers (names) to use for events created by the check. Each array item must be a string.
required	false
type	Array
example	`"handlers": ["pagerduty", "email"]`

interval

description	How often the check is executed. In seconds.
required	true (unless `cron` is configured)
type	Integer

example	
	```"interval": 60```

cron

description	When the check should be executed, using <u>cron syntax</u> or <u>these predefined schedules</u>. 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 (e.g.* `cron: '* * * * *'` *).*
required	true (unless `interval` is configured)
type	String
example	```"cron": "0 0 * * *"```

publish

description	`true` if check requests are published for the check. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	```"publish": false```

timeout

description	Check execution duration timeout (hard stop). In seconds.

required	false
type	Integer
example	`"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	`"ttl": 100`

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	`"stdin": true`

low_flap_threshold

description	Flap detection low threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios.
required	false
type	Integer
example	`"low_flap_threshold": 20`

high_flap_threshold

description	Flap detection high threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios.
required	true (if `low_flap_threshold` is configured)
type	Integer
example	`"high_flap_threshold": 60`

runtime_assets

description	Array of Sensu assets (names). Required at runtime for the execution of the `command`.
required	false
type	Array
example	`"runtime_assets": ["ruby-2.5.0"]`

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	

```
"check_hooks": [
  {
    "0": [
      "passing-hook","always-run-this-hook"
    ]
  },
  {
    "critical": [
      "failing-hook","collect-diagnostics","always-run-
this-hook"
    ]
  }
]
```

proxy_entity_name

description	Entity name. Used to create a proxy entity for an external resource (e.g. a network switch).
required	false
type	String
validated	`\A[\w\.\-]+\z`
example	`"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`. Proxy requests are a great way to reuse check definitions for a group of entities. For more information, see the proxy requests specification and Monitor external resources.
required	false
type	Hash
example	(see below)

```
"proxy_requests": {
  "entity_attributes": [
    "entity.entity_class == 'proxy'",
    "entity.labels.proxy_type == 'website'"
  ],
  "splay": true,
  "splay_coverage": 90
}
```

silenced

description	Silences that apply to the check.
type	Array
example	```"silenced": ["*:routers"]```

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
example	```"env_vars": ["RUBY_VERSION=2.5.0", "CHECK_HOST=my.host.internal"]```

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) 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

example	
	`"output_metric_format": "graphite_plaintext"`

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. For an example, see the Sensu InfluxDB handler.
required	false
type	Array
example	`"output_metric_handlers": ["influx-db"]`

round_robin

description	When set to `true`, Sensu executes the check once per interval, cycling through each subscribing agent in turn. See 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. See 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	`"round_robin": true`

subdue	
description	Check subdues are not yet implemented in Sensu Go. Although the `subdue` attribute appears in check definitions by default, it is a placeholder and should not be modified.
example	`"subdue": null`

Proxy requests attributes

entity_attributes	
description	Sensu entity attributes to match entities in the registry using Sensu query expressions.
required	false
type	Array
example	```"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`. For example, if a check has an interval of `60` seconds and a configured splay coverage of `90` %, its proxy check requests would be splayed evenly over a time window of `60` seconds * `90` %, `54` seconds, leaving `6` seconds for the last proxy check execution before the the next round of proxy check requests for the same check.

required	false
type	Boolean
default	`false`
example	`"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 amount of time check requests can be published over (before the next check interval).
required	Required if `splay` attribute is set to `true`
type	Integer
example	`"splay_coverage": 90`

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.

required	false
type	Integer
example	`"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	`"discard_output": true`

Examples

Minimum recommended check attributes

NOTE: The attribute `interval` is not required if a valid `cron` schedule is defined.

YML

```yml
type: CheckConfig
api_version: core/v2
metadata:
  name: check_minimum
  namespace: default
spec:
  command: collect.sh
  handlers:
  - slack
```

```
  interval: 10
  publish: true
  subscriptions:
  - system
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "check_minimum"
  },
  "spec": {
    "command": "collect.sh",
    "subscriptions": [
      "system"
    ],
    "handlers": [
      "slack"
    ],
    "interval": 10,
    "publish": true
  }
}
```

Metric check

YML

```yaml
type: CheckConfig
api_version: core/v2
metadata:
  annotations:
    slack-channel: '#monitoring'
  labels:
    region: us-west-1
  name: collect-metrics
  namespace: default
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: graphite_plaintext
output_metric_handlers:
- influx-db
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": {
    "name": "collect-metrics",
    "namespace": "default",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "slack-channel" : "#monitoring"
    }
  },
  "spec": {
    "command": "collect.sh",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
```

```
      "publish": true,
      "runtime_assets": null,
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "output_metric_format": "graphite_plaintext",
      "output_metric_handlers": [
        "influx-db"
      ],
      "env_vars": null,
      "discard_output": true
    }
  }
```

PowerShell script in check commands

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
  namespace: default
spec:
  command: powershell.exe -f c:\\users\\tester\\test.ps1
  subscriptions:
  - system
  handlers:
  - slack
  interval: 60
  publish: true
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "interval_test",
    "namespace": "default"
  },
  "spec": {
    "command": "powershell.exe -f c:\\users\\tester\\test.ps1",
    "subscriptions": ["system"],
    "handlers": ["slack"],
    "interval": 60,
    "publish": true
  }
}
```

Datastore

Event storage

Sensu stores the most recent event for each entity and check pair using either an embedded etcd (default) or an external etcd instance. You can access event data with the Sensu dashboard Events page, `sensuctl event` commands, and the events API. For longer retention of event data, integrate Sensu with a time series database like InfluxDB or a searchable index like ElasticSearch or Splunk.

Scale event storage

COMMERCIAL FEATURE: Access enterprise-scale event storage in the packaged Sensu Go distribution. For more information, see 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 8-GB 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.

Requirements

Sensu supports PostgreSQL 9.5 and later, including Amazon Relational Database Service (Amazon RDS) when configured with the PostgreSQL engine. See the PostgreSQL docs to install and configure

PostgreSQL.

Configuration

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. See Datastore specification for more information.

YML

```
type: PostgresConfig
api_version: store/v1
metadata:
  name: my-postgres
spec:
  dsn: "postgresql://user:secret@host:port/dbname"
  pool_size: 20
```

JSON

```
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "dsn": "postgresql://user:secret@host:port/dbname",
    "pool_size": 20
```

```
    }
  }
```

With the `PostgresConfig` resource definition saved to a file (for example, `postgres.yml`), use sensuctl, configured as the admin user, to activate the PostgreSQL event store.

```
sensuctl create --file postgres.yml
```

To update your Sensu PostgreSQL configuration, repeat the `sensuctl create` process. You can expect to see 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.

Disable the PostgreSQL event store

To disable the PostgreSQL event store, use `sensuctl delete` with your `PostgresConfig` resource definition:

```
sensuctl delete --file postgres.yml
```

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

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 | ```
type: PostgresConfig
``` |

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 | ```
api_version: store/v1
``` |

metadata

description	Top-level scope that contains the PostgreSQL datastore `name`.
required	true
type	Map of key-value pairs
| example | ```
metadata:
  name: my-postgres
``` |

spec	
description	Top-level map that includes the PostgreSQL datastore config spec attributes.
required	true
type	Map of key-value pairs
| example | ```yaml
spec:
  dsn: "postgresql://user:secret@host:port/dbname"
  pool_size: 20
``` |

Metadata attributes

name	
description	PostgreSQL datastore name used internally by Sensu.
required	true
type	String
| example | ```yaml
name: my-postgres
``` |

Spec attributes

dsn	
description	Data source names. Specified as a URL or PostgreSQL connection string. See the PostgreSQL docs for more information about connection strings.

required	true
type	String
| example | ```
dsn: "postgresql://user:secret@host:port/dbname"
``` |

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 | ```
pool_size: 20
``` |

Entities

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 (from server hardware to serverless functions). We call these monitored parts of an infrastructure "entities."

An entity provides context for event data — what and where the event is from — and an event's uniqueness is determined by the check name and the name of the entity upon which the check ran. Entities can also contain system information like the hostname, operating system, platform, and version.

Agent entities are monitoring agents that are installed and run on every system that needs to be monitored. The agent entity registers the system with the Sensu backend service, sends keepalive messages (the Sensu heartbeat mechanism), and executes monitoring checks. Each entity is a member of one or more `subscriptions` : a list of roles and responsibilities assigned to the agent entity (e.g. 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).

Proxy entities are dynamically created entities that are added 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 a Sensu agent cannot be installed (like a network switch or website) using the defined check `ProxyEntityName` to create a proxy entity for the external resource.

Usage limits

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

Proxy entities

Proxy entities [formerly known as proxy clients or just-in-time (JIT) clients] are dynamically created entities that are added 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 and devices where a Sensu agent cannot be installed (like a network switch or website) using the defined check `ProxyEntityName` to create a proxy entity for the external resource.

Proxy entity registration differs from keepalive-based registration because the registration event happens while processing a check result (not a keepalive message).

See Monitor external resources to learn how to use a proxy entity to monitor a website.

Proxy entities and round robin scheduling

Proxy entities make round robin scheduling more useful. Proxy entities 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 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. If you don't create a proxy entity, it is created when the check is executed. You can modify the proxy entity later if needed.

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.

Manage entity labels

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

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.

Proxy entity labels

For entities with class `proxy`, you can create and manage labels with sensuctl. For example, to create a proxy entity with a `url` label using sensuctl `create`, create a file called `example.json` with an entity definition that includes `labels`:

YML

```yml
type: Entity
api_version: core/v2
sensu_agent_version: 1.0.0
metadata:
  labels:
    url: docs.sensu.io
  name: sensu-docs
  namespace: default
spec:
  deregister: false
  deregistration: {}
  entity_class: proxy
  last_seen: 0
  subscriptions:
  - proxy
  system:
    network:
      interfaces: null
```

JSON

```json
{
  "type": "Entity",
  "api_version": "core/v2",
  "sensu_agent_version": "1.0.0",
  "metadata": {
```

```json
      "name": "sensu-docs",
      "namespace": "default",
      "labels": {
        "url": "docs.sensu.io"
      }
    },
    "spec": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "proxy",
      "last_seen": 0,
      "subscriptions": [
        "proxy"
      ],
      "system": {
        "network": {
          "interfaces": null
        }
      }
    }
  }
```

NOTE: *The proxy entity definition must include the same subscriptions as the sensu-agent to work with round robin scheduling **and** <u>proxy requests attributes</u>. If more than one sensu-agent will execute a proxy check and you did not configure the proxy entity with the same subscriptions as the sensu-agent, the sensu-backend will log an error and the proxy check will not be scheduled for agents to run.*

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

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

To add a label to an existing entity, use sensuctl `edit` .For example, run `sensuctl edit` to add a `url` label to a `sensu-docs` entity:

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

And update the `metadata` scope to include `labels`:

YML

```yml
type: Entity
api_version: core/v2
sensu_agent_version: 1.0.0
metadata:
  labels:
    url: docs.sensu.io
  name: sensu-docs
  namespace: default
spec:
  '...': '...'
```

JSON

```json
{
  "type": "Entity",
  "api_version": "core/v2",
  "sensu_agent_version": "1.0.0",
  "metadata": {
    "name": "sensu-docs",
    "namespace": "default",
    "labels": {
      "url": "docs.sensu.io"
    }
  },
  "spec": {
    "...": "..."
  }
}
```

Proxy entity checks

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 check with a proxy entity name to associate the check results with that proxy entity. On the first check result, if the proxy entity does not exist, Sensu will create the entity as a proxy entity.

After you create a proxy entity check, define which agents will run the check by configuring a

subscription.See proxy entities for details about creating a proxy check for a proxy entity.

Agent entity labels

For 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
```

Entities specification

Top-level attributes

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	`"type": "Entity"`

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
example	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the entity, including the `name` and `namespace` 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. See <u>metadata attributes</u> for details.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs
example	(see code below)

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

sensu_agent_version

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

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 | ```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"
            ]
          },
          {
``` |

```json
          "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"
  },
  "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"
  ]
}
```

Metadata attributes

name

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

namespace

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

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering and and tokens. If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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 filtering. For complex, non-identifying metadata that you will *not* need to use for API response, sensuctl, or dashboard view 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 | ```
"labels": {
  "environment": "development",
  "region": "us-west-2"
}
``` |

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
| example | ```
"annotations": {
  "managed-by": "ops",
  "playbook": "www.example.url"
}
``` |

Spec attributes

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. 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
example	`"entity_class": "agent"`

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. If you are using round robin scheduling **and** proxy requests attributes, the proxy entity definition must include the same subscriptions as the sensu-agent.
required	false
type	Array
default	The entity-specific subscription.
example	`"subscriptions": ["web", "prod", "entity:example-entity"]`

system

description	System information about the entity, such as operating system and platform. See system attributes for more information.
required	false

type	Map

YML

```yaml
system:
  arch: amd64
  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

```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"
         }
       }
```

last_seen

description	Timestamp the entity was last seen. In seconds since the Unix epoch.
required	false
type	Integer
example	`"last_seen": 1522798317`

deregister

description	`true` if the entity should be removed when it stops sending keepalive messages. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	`"deregister": false`

deregistration

description	Map that contains a handler name to use when an entity is deregistered. See deregistration attributes for more information.
required	false
type	Map

YML

```yaml
deregistration:
  handler: email-handler
```

JSON

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

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"]

YML

```yaml
redact:
- extra_secret_tokens
```

JSON

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

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	`"user": "agent"`

System attributes

hostname

description	Hostname of the entity.
required	false
type	String
example	`"hostname": "example-hostname"`

os

description	Entity's operating system.
required	false
type	String
example	`"os": "linux"`

platform

description	Entity's operating system distribution.
required	false
type	String
example	`"platform": "ubuntu"`

platform_family

description	Entity's operating system family.
required	false
type	String
example	`"platform_family": "debian"`

platform_version

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

```
"platform_version": "16.04"
```

network

description	Entity's network interface list. See network attributes for more information.
required	false
type	Map

YML

example	

```yaml
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

```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"
          ]
        }
      ]
    }
  }
```

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
example	`"arch": "amd64"`

Network attributes

network_interface	
description	List of network interfaces available on the entity, with their associated MAC and IP addresses.
required	false

type	Array NetworkInterface

YML

```yaml
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

```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"
      ]
    }
  ]
}
```

NetworkInterface attributes

name

description	Network interface name.
required	false
type	String
example	`"name": "eth0"`

mac

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

addresses

description	List of IP addresses for the network interface.
required	false
type	Array
example	`"addresses": ["93.184.216.34/24", "2606:2800:220:1:248:1893:25c8:1946/10"]`

Deregistration attributes

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

Examples

Entity definition
YML

```yaml
type: Entity
api_version: core/v2
sensu_agent_version: 1.0.0
metadata:
  annotations: null
  labels: null
  name: webserver01
  namespace: default
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
      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
    user: agent
```

JSON

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "sensu_agent_version": "1.0.0",
  "metadata": {
    "name": "webserver01",
    "namespace": "default",
    "labels": null,
    "annotations": null
```

```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"
        },
        "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"
      ]
    }
  }
```

Etcd replicators

COMMERCIAL FEATURE: Access the etcd-replicators datatype in the packaged Sensu Go distribution.For more information, see Get started with commercial features.

NOTE: etcd-replicators is a datatype in the federation API, which is only accessible for users who have a cluster role that permits access to replication resources.

Etcd replicators allow you to manage RBAC resources in one place and mirror the changes to follower clusters.The API sets up etcd mirrors for one-way key replication.

The etcd-replicators datatype will not use a namespace because it applies cluster-wide.Therefore, only cluster role RBAC bindings will apply to it.

Create a replicator

You can use `sensuctl create` or the Sensu web UI 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-replicators specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Always `EtcdReplicator.`
required	true
type	String
| example | ```
type: EtcdReplicator
``` |

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	

```
api_version: federation/v1
```

metadata

description	Top-level scope that contains the replicator `name`. Namespace is not supported in the metadata because etcd replicators are cluster-wide resources.
required	true
type	Map of key-value pairs
example	

```
metadata:
  name: my_replicator
```

spec

description	Top-level map that includes the replicator spec attributes.
required	true
type	Map of key-value pairs
example	

```
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
```

Metadata attributes

name	
description	Replicator name used internally by Sensu.
required	true
type	String
example	`name: my_replicator`

Spec attributes

ca_cert	
description	Path to an the PEM-format CA certificate to use for TLS client authentication.
required	true if `insecure: false` (which is the default configuration). If `insecure: true`, `ca_cert` is not required.
type	String
example	`ca_cert: /path/to/trusted-certificate-authorities.pem`

cert

description	Path to the PEM-format certificate to use for TLS client authentication.
required	true if `insecure: false` (which is the default configuration). If `insecure: true`, `cert` is not required.
type	String
| example | ```
cert: /path/to/ssl/cert.pem
``` |

key

description	Path to the PEM-format key file associated with the `cert` to use for TLS client authentication.
required	true if `insecure: false` (which is the default configuration). If `insecure: true`, `key` is not required.
type	String
| example | ```
key: /path/to/ssl/key.pem
``` |

insecure

description	`true` to disable transport security. Otherwise, `false`. *NOTE: Disable transport security with care.*
required	false
type	Boolean
default	`false`
| example | ```
insecure: false
``` |

url

description	Destination cluster URL. If specifying more than one, use a comma to separate.
required	true
type	String
example	

```
url: http://127.0.0.1:2379
```

api_version

description	Sensu API version of the resource to replicate.
required	false
type	String
default	`core/v2`
example	

```
api_version: core/v2
```

resource

description	Name of the resource to replicate.
required	true
type	String
example	

```
resource: Role
```

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	`namespace: default`

replication_interval _seconds	
description	Interval at which the resource will be replicated. In seconds.
required	false
type	String
default	30
example	`replication_interval_seconds: 30`

Example EtcdReplicator resources

If you replicate the following four examples for `Role`, `RoleBinding`, `ClusterRole`, and `ClusterRoleBinding` resources, you can expect 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.

Example `Role` resource

YML

```yml
api_version: federation/v1
type: EtcdReplicator
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

```json
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "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
  }
}
```

Example `RoleBinding` resource

YML

```yml
api_version: federation/v1
```

```
type: EtcdReplicator
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

```
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "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
  }
}
```

Example `ClusterRole` resource
YML

```
api_version: federation/v1
type: EtcdReplicator
metadata:
  name: clusterrole_replicator
```

```yaml
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

```json
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "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
  }
}
```

Example `ClusterRoleBinding` resource

YML

```yaml
api_version: federation/v1
type: EtcdReplicator
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

```
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "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.

Events

An event is a generic container used by Sensu to provide context to checks and metrics. The context, called event data, contains information about the originating entity and the corresponding check or metric result. An event must contain a check or metrics. In certain cases, an event can contain both. These generic containers allow Sensu to handle different types of events in the pipeline. Because events are polymorphic in nature, it is important to never assume their contents (or lack of content).

Check-only events

A Sensu event is created every time a check result is processed by the Sensu server, regardless of the status indicated by the check result. The agent creates an event upon receipt of the check execution result. The agent will execute any configured hooks the check might have. From there, the result is forwarded to the Sensu backend for processing. Potentially noteworthy events may be processed by one or more event handlers, for example to send an email or invoke an automated action.

Metric-only events

Sensu events can also 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.

Check and metric 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.

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 using the events API

You can send events directly to the Sensu pipeline using the events API. To create an event, send a JSON event definition to the events API PUT endpoint.

Manage events

You can manage events using the Sensu dashboard, events API, and sensuctl command line tool.

View events

To list all events:

```
sensuctl event list
```

To show event details in the default <u>output format</u>:

```
sensuctl event info entity-name check-name
```

With both the `list` and `info` commands, you can specify an <u>output format</u> using the `--format` flag:

- ⌐ `yaml` or `wrapped-json` formats for use with <u>sensuctl create</u>
- ⌐ `json` format for use with the <u>events API</u>

```
sensuctl event info entity-name check-name --format yaml
```

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 see 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 see a confirmation message upon success:

```
Resolved
```

Event format

Sensu events contain:

- `entity` scope (required)

    - Information about the source of the event, including any attributes defined in the <u>entity specification</u>

- `check` scope (optional if the `metrics` scope is present)

    - Information about how the event was created, including any attributes defined in the <u>check specification</u>

    - Information about the event and its history, including any check attributes defined in the <u>event specification on this page</u>

- `metrics` scope (optional if the `check` scope is present)

    - Metric points in <u>Sensu metric format</u>

- `timestamp`

    - Time that the event occurred in seconds since the Unix epoch

Use event data

Event data is a powerful tool for automating monitoring workflows. For example, you can reduce alert fatigue by filtering events based on the event `occurrences` attribute.

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`

Events specification

Top-level attributes

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	`"type": "Event"`

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	`"api_version": "core/v2"`

metadata

description	Top-level scope that contains the event `namespace`. 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. See the <u>metadata attributes</u> for details.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

example	<pre>"metadata": {
	  "namespace": "default"
	}</pre>

spec

description	Top-level map that includes the event <u>spec attributes</u>.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

example	<pre>"spec": {
	  "check": {
	    "check_hooks": null,
	    "command": "/opt/sensu-plugins-
	ruby/embedded/bin/metrics-curl.rb -u \"http://localhost\"",
	    "duration": 0.060790838,
	    "env_vars": null,
	    "executed": 1552506033,
	    "handlers": [],
	    "high_flap_threshold": 0,
	    "history": [
	      {</pre>

```
      "executed": 1552505833,
      "status": 0
    },
    {
      "executed": 1552505843,
      "status": 0
    }
  ],
  "interval": 10,
  "issued": 1552506033,
  "last_ok": 1552506033,
  "low_flap_threshold": 0,
  "metadata": {
    "name": "curl_timings",
    "namespace": "default"
  },
  "occurrences": 1,
  "occurrences_watermark": 1,
  "output": "sensu-go-sandbox.curl_timings.time_total
0.005 1552506033\nsensu-go-
sandbox.curl_timings.time_namelookup 0.004",
  "output_metric_format": "graphite_plaintext",
  "output_metric_handlers": [
    "influx-db"
  ],
  "proxy_entity_name": "",
  "publish": true,
  "round_robin": false,
  "runtime_assets": [],
  "state": "passing",
  "status": 0,
  "stdin": false,
  "subdue": null,
  "subscriptions": [
    "entity:sensu-go-sandbox"
  ],
  "timeout": 0,
  "total_state_change": 0,
  "ttl": 0
},
"entity": {
  "deregister": false,
```

```json
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1552495139,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "subscriptions": [
        "entity:sensu-go-sandbox"
      ],
      "system": {
        "arch": "amd64",
        "hostname": "sensu-go-sandbox",
        "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"
  },
  "metrics": {
    "handlers": [
      "influx-db"
    ],
    "points": [
      {
        "name": "sensu-go-sandbox.curl_timings.time_total",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "sensu-go-
sandbox.curl_timings.time_namelookup",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.004
      }
    ]
  },
  "timestamp": 1552506033
}
```

Metadata attributes

namespace	
description	Sensu RBAC namespace that this event belongs to.
required	false

type	String
default	`default`
example	`"namespace": "production"`

Spec attributes

timestamp	
description	Time that the event occurred. In seconds since the Unix epoch.
required	false
type	Integer
default	Time that the event occurred
example	`"timestamp": 1522099512`

entity	
description	Entity attributes from the originating entity (agent or proxy).
type	Map
required	true
example	

```
"entity": {
  "deregister": false,
  "deregistration": {},
  "entity_class": "agent",
  "last_seen": 1552495139,
  "metadata": {
    "name": "sensu-go-sandbox",
```

```json
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "subscriptions": [
      "entity:sensu-go-sandbox"
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-go-sandbox",
      "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"
  }
```

check

description	<u>Check definition</u> used to create the event and information about the status and history of the event. The check scope includes attributes described in the <u>event specification</u> and the <u>check specification</u>.
type	Map
required	true
example	

```
"check": {
  "check_hooks": null,
  "command": "/opt/sensu-plugins-ruby/embedded/bin/metrics-
curl.rb -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,
  "issued": 1552506033,
  "last_ok": 1552506033,
  "low_flap_threshold": 0,
  "metadata": {
    "name": "curl_timings",
```

```
      "namespace": "default"
    },
    "occurrences": 1,
    "occurrences_watermark": 1,
    "output": "sensu-go-sandbox.curl_timings.time_total
0.005",
    "output_metric_format": "graphite_plaintext",
    "output_metric_handlers": [
      "influx-db"
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [],
    "state": "passing",
    "status": 0,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "entity:sensu-go-sandbox"
    ],
    "timeout": 0,
    "total_state_change": 0,
    "ttl": 0
  }
```

metrics	
description	Metrics collected by the entity in Sensu metric format. See the metric attributes.
type	Map
required	false
example	```"metrics": {
"handlers": [
"influx-db"
],
``

```
      "points": [
        {
          "name": "sensu-go-sandbox.curl_timings.time_total",
          "tags": [],
          "timestamp": 1552506033,
          "value": 0.005
        },
        {
          "name": "sensu-go-
sandbox.curl_timings.time_namelookup",
          "tags": [],
          "timestamp": 1552506033,
          "value": 0.004
        }
      ]
    }
```

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
example	`"duration": 1.903135228`

executed	
description	Time at which the check request was executed. In seconds since the

Unix epoch.

required	false
type	Integer
example	``` "executed": 1522100915 ```

history

description	Check status history for the last 21 check executions. See history attributes.
required	false
type	Array

example	```
	"history": [
	  {
	    "executed": 1552505983,
	    "status": 0
	  },
	  {
	    "executed": 1552505993,
	    "status": 0
	  }
	]
	``

issued

description	Time that the check request was issued. In seconds since the Unix epoch.
required	false
type	Integer

example	
| | ```
"issued": 1552506033
``` |

last_ok

description	Last time that the check returned an OK status (`0`). In seconds since the Unix epoch.
required	false
type	Integer
| example | ```
"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. See Use event data for more information.
required	false
type	Integer greater than 0
| example | ```
"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` . See Use event
data for more information.

required	false
type	Integer greater than 0
example	`"occurrences_watermark": 1`

output

description	Output from the execution of the check command.
required	false
type	String
example	`"output": "sensu-go-sandbox.curl_timings.time_total 0.005"`

state

description	State of the check: `passing` (status `0`), `failing` (status other than `0`), or `flapping` . You can use the `low_flap_threshold` and `high_flap_threshold` check attributes to configure `flapping` state detection.
required	false
type	String

example	
	`"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.
required	false
type	Integer
example	`"status": 0`

total_state_change

description	Total state change percentage for the check's history.
required	false
type	Integer
example	`"total_state_change": 0`

History attributes

executed	
description	Time at which the check request was executed. In seconds since the Unix epoch.
required	false
type	Integer
example	```"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.

required	false
type	Integer
example	```"status": 0```

Metric 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	

```
"handlers": [
  "influx-db"
]
```

points

description	Metric data points, including a name, timestamp, value, and tags. See points attributes.
required	false
type	Array
example	

```
"points": [
  {
    "name": "sensu-go-sandbox.curl_timings.time_total",
    "tags": [
      {
        "name": "response_time_in_ms",
        "value": "101"
      }
    ],
    "timestamp": 1552506033,
    "value": 0.005
  },
  {
    "name": "sensu-go-
sandbox.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
example	```"name": "sensu-go-sandbox.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
example	```"tags": [``` ``` {``` ``` "name": "response_time_in_ms",``` ``` "value": "101"``` ``` }``` ```]```

timestamp

description	Time at which the metric was collected. In seconds since the Unix epoch.
required	false
type	Integer
example	

```
"timestamp": 1552506033
```

value

description	Metric value.
required	false
type	Float
example	

```
"value": 0.005
```

Examples

Example check-only event data

YML

```yaml
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
    command: check-cpu.sh -w 75 -c 90
```

```yaml
    duration: 1.07055808
    env_vars: null
    executed: 1552594757
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1552594757
      status: 0
    interval: 60
    issued: 1552594757
    last_ok: 1552594758
    low_flap_threshold: 0
    metadata:
      name: check-cpu
      namespace: default
    occurrences: 1
    occurrences_watermark: 1
    output: |
      CPU OK - Usage:3.96
    output_metric_format: ""
    output_metric_handlers: []
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets: []
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - linux
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1552594641
    metadata:
      name: sensu-centos
      namespace: default
```

```
      redact:
      - password
      - passwd
      - pass
      - api_key
      - api_token
      - access_key
      - secret_key
      - private_key
      - secret
      subscriptions:
      - linux
      - entity:sensu-centos
      system:
        arch: amd64
        hostname: sensu-centos
        network:
          interfaces:
          - addresses:
            - 127.0.0.1/8
            - ::1/128
            name: lo
          - addresses:
            - 10.0.2.15/24
            - fe80::9688:67ca:3d78:ced9/64
            mac: 08:00:27:11:ad:d2
            name: enp0s3
          - addresses:
            - 172.28.128.3/24
            - fe80::a00:27ff:fe6b:c1e9/64
            mac: 08:00:27:6b:c1:e9
            name: enp0s8
        os: linux
        platform: centos
        platform_family: rhel
        platform_version: 7.4.1708
      user: agent
    timestamp: 1552594758
```

JSON

```
  {
```

```json
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "check-cpu.sh -w 75 -c 90",
      "duration": 1.07055808,
      "env_vars": null,
      "executed": 1552594757,
      "handlers": [],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1552594757,
          "status": 0
        }
      ],
      "interval": 60,
      "issued": 1552594757,
      "last_ok": 1552594758,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      },
      "occurrences": 1,
      "occurrences_watermark": 1,
      "output": "CPU OK - Usage:3.96\n",
      "output_metric_format": "",
      "output_metric_handlers": [],
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [],
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
```

```json
      "linux"
    ],
    "timeout": 0,
    "total_state_change": 0,
    "ttl": 0
  },
  "entity": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "agent",
    "last_seen": 1552594641,
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "subscriptions": [
      "linux",
      "entity:sensu-centos"
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-centos",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ],
            "name": "lo"
          },
```

```
        {
          "addresses": [
            "10.0.2.15/24",
            "fe80::9688:67ca:3d78:ced9/64"
          ],
          "mac": "08:00:27:11:ad:d2",
          "name": "enp0s3"
        },
        {
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe6b:c1e9/64"
          ],
          "mac": "08:00:27:6b:c1:e9",
          "name": "enp0s8"
        }
      ]
    },
    "os": "linux",
    "platform": "centos",
    "platform_family": "rhel",
    "platform_version": "7.4.1708"
  },
  "user": "agent"
},
"timestamp": 1552594758
  }
}
```

Example event with check and metric data

YML

```
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
    command: /opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -u
"http://localhost"
```

```yaml
    duration: 0.060790838
    env_vars: null
    executed: 1552506033
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1552505833
      status: 0
    - executed: 1552505843
      status: 0
    interval: 10
    issued: 1552506033
    last_ok: 1552506033
    low_flap_threshold: 0
    metadata:
      name: curl_timings
      namespace: default
    occurrences: 1
    occurrences_watermark: 1
    output: |-
      sensu-go-sandbox.curl_timings.time_total 0.005 1552506033
      sensu-go-sandbox.curl_timings.time_namelookup 0.004
    output_metric_format: graphite_plaintext
    output_metric_handlers:
    - influx-db
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets: []
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - entity:sensu-go-sandbox
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
```

```yaml
      last_seen: 1552495139
    metadata:
      name: sensu-go-sandbox
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
    subscriptions:
    - entity:sensu-go-sandbox
    system:
      arch: amd64
      hostname: sensu-go-sandbox
      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
  metrics:
    handlers:
    - influx-db
    points:
    - name: sensu-go-sandbox.curl_timings.time_total
      tags: []
      timestamp: 1552506033
```

```
      value: 0.005
    - name: sensu-go-sandbox.curl_timings.time_namelookup
      tags: []
      timestamp: 1552506033
      value: 0.004
  timestamp: 1552506033
```

JSON

```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "/opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -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,
      "issued": 1552506033,
      "last_ok": 1552506033,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "curl_timings",
        "namespace": "default"
```

```
      },
      "occurrences": 1,
      "occurrences_watermark": 1,
      "output": "sensu-go-sandbox.curl_timings.time_total 0.005 1552506033\nsensu-
  go-sandbox.curl_timings.time_namelookup 0.004",
      "output_metric_format": "graphite_plaintext",
      "output_metric_handlers": [
        "influx-db"
      ],
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [],
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "entity:sensu-go-sandbox"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
    },
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1552495139,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
```

```
          "secret"
        ],
        "subscriptions": [
          "entity:sensu-go-sandbox"
        ],
        "system": {
          "arch": "amd64",
          "hostname": "sensu-go-sandbox",
          "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"
      },
      "metrics": {
        "handlers": [
          "influx-db"
        ],
        "points": [
          {
            "name": "sensu-go-sandbox.curl_timings.time_total",
            "tags": [],
```

```
      "timestamp": 1552506033,
      "value": 0.005
    },
    {
      "name": "sensu-go-sandbox.curl_timings.time_namelookup",
      "tags": [],
      "timestamp": 1552506033,
      "value": 0.004
    }
  ]
 },
 "timestamp": 1552506033
 }
}
```

Example metric-only event

YML

```
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1552495139
    metadata:
      name: sensu-go-sandbox
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
```

```
      subscriptions:
      - entity:sensu-go-sandbox
      system:
        arch: amd64
        hostname: sensu-go-sandbox
        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
    metrics:
      handlers:
      - influx-db
      points:
      - name: sensu-go-sandbox.curl_timings.time_total
        tags: []
        timestamp: 1552506033
        value: 0.005
      - name: sensu-go-sandbox.curl_timings.time_namelookup
        tags: []
        timestamp: 1552506033
        value: 0.004
    timestamp: 1552506033
```

JSON

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

```json
        "namespace": "default"
      },
      "spec": {
        "entity": {
          "deregister": false,
          "deregistration": {},
          "entity_class": "agent",
          "last_seen": 1552495139,
          "metadata": {
            "name": "sensu-go-sandbox",
            "namespace": "default"
          },
          "redact": [
            "password",
            "passwd",
            "pass",
            "api_key",
            "api_token",
            "access_key",
            "secret_key",
            "private_key",
            "secret"
          ],
          "subscriptions": [
            "entity:sensu-go-sandbox"
          ],
          "system": {
            "arch": "amd64",
            "hostname": "sensu-go-sandbox",
            "network": {
              "interfaces": [
                {
                  "addresses": [
                    "127.0.0.1/8",
                    "::1/128"
                  ],
                  "name": "lo"
                },
                {
                  "addresses": [
                    "10.0.2.15/24",
                    "fe80::5a94:f67a:1bfc:a579/64"
```

```json
        ],
        "mac": "08:00:27:8b:c9:3f",
        "name": "eth0"
      }
    ]
  },
  "os": "linux",
  "platform": "centos",
  "platform_family": "rhel",
  "platform_version": "7.5.1804"
},
"user": "agent"
},
"metrics": {
  "handlers": [
    "influx-db"
  ],
  "points": [
    {
      "name": "sensu-go-sandbox.curl_timings.time_total",
      "tags": [],
      "timestamp": 1552506033,
      "value": 0.005
    },
    {
      "name": "sensu-go-sandbox.curl_timings.time_namelookup",
      "tags": [],
      "timestamp": 1552506033,
      "value": 0.004
    }
  ]
},
"timestamp": 1552506033
}
}
```

Filters

Sensu event filters are applied when you configure event handlers to use one or more filters.Before executing a handler, the Sensu backend will apply any event filters configured for the handler to the event data.If the filters do not remove the event, the handler will be executed.

The filter analysis performs these steps:

- ⌐ When the Sensu backend is processing an event, it checks for the definition of a `handler` (or `handlers`).Before executing each handler, the Sensu server first applies any configured `filters` for the handler.

- ⌐ If multiple `filters` are configured for a handler, 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).

As soon as a filter removes an event, no further analysis is performed and the event handler will not be executed.

*NOTE: Filters specified in a **handler set** definition have no effect. Filters must be specified in individual handler definitions.*

Inclusive and exclusive event filters

Event filters can be *inclusive* (`"action": "allow"` ; replaces `"negate": false` in Sensu Core) or *exclusive* (`"action": "deny"` ; replaces `"negate": true` in Sensu Core).Configuring a handler to use multiple *inclusive* event filters is the equivalent of using an `AND` query operator (only handle events if they match the *inclusive* filter: `x AND y AND z`).Configuring a handler to use multiple *exclusive* event filters is the equivalent of using an `OR` operator (only handle events if they don't match `x OR y OR z`).

In **inclusive filtering**, by setting the event filter definition attribute `"action": "allow"` , only events that match the defined filter expressions are handled.

In **exclusive filtering**, by setting the event filter definition attribute `"action": "deny"` , events are only handled if they do not match the defined filter expressions.

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.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.Event filters that return a `true` value will continue to be processed via additional filters (if defined), mutators (if defined), and handlers.

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.There are some caveats to using Otto: not all of the regular expressions specified in ECMAScript 5 will work.See the Otto README for more details.

Filter assets

Sensu event filters can have assets that are included in their execution context.When valid 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, see 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, see Send Slack alerts and Plan maintenance.

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 handler, the is_incident filter allows only warning (`"status": 1`), critical (`"status": 2`), and resolution events to be processed.

To use the is_incident event filter, include `is_incident` in the handler configuration `filters` array:

YML

```yaml
type: Handler
api_version: core/v2
metadata:
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXX
  filters:
  - is_incident
  handlers: []
  runtime_assets: []
  timeout: 0
  type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXXX"
    ],
    "filters": [
      "is_incident"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
}
```

The is_incident event filter applies the following filtering logic:

status	allow	discard
0		❌
1	✅	
2	✅	
other		❌
1 –> 0 or 2 –> 0 (resolution event)	✅	

Built-in filter: not_silenced

Sensu silencing lets you suppress execution of event handlers on an on-demand basis so you can quiet incoming alerts and plan maintenance.

To allow silencing for an event handler, add `not_silenced` to the handler configuration `filters` array:

YML

```yaml
type: Handler
api_version: core/v2
metadata:
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXX
  filters:
  - is_incident
  - not_silenced
  handlers: []
  runtime_assets: []
  timeout: 0
  type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [
```

```
  "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
  XXXXXXXXXXXX"
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
}
```

When applied to a handler configuration, the not_silenced event filter silences events that include the `silenced` attribute. The handler 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 to a handler, 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 handler configuration `filters` array:

YML

```
type: Handler
api_version: core/v2
metadata:
  name: influx-db
  namespace: default
spec:
  command: sensu-influxdb-handler -d sensu
  env_vars:
  - INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086
  - INFLUXDB_USER=sensu
```

```
    - INFLUXDB_PASSWORD=password
  filters:
  - has_metrics
  handlers: []
  runtime_assets: []
  timeout: 0
  type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "influx-db",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-influxdb-handler -d sensu",
    "env_vars": [
      "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
      "INFLUXDB_USER=sensu",
      "INFLUXDB_PASSWORD=password"
    ],
    "filters": [
      "has_metrics"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
}
```

When applied to a handler configuration, the has_metrics event filter allows only events that include a
`metrics` scope.

Build event filter 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, see 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_inciden`	B	Returns true for critical alerts (status `2`), warnings (status `1`),

attribute	type	description
`t`	Boolean	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`	st	Check execution schedule using cron syntax

	ri n g	
`event.check.disca` `rd_output`	B o ol e a n	Whether the check is configured to discard check output from event data
`event.check.durat` `ion`	fl o at	Command execution time in seconds
`event.check.env_v` `ars`	ar ra y	Environment variables used with command execution
`event.check.execu` `ted`	in te g er	Time that the check was executed in seconds since the Unix epoch
`event.check.handl` `ers`	ar ra y	Sensu event <u>handlers</u> assigned to the check
`event.check.high_` `flap_threshold`	in te g er	The check's flap detection high threshold in percent state change
`event.check.histo` `ry`	ar ra y	<u>Check status history</u> for the last 21 check executions
`event.check.hook` `s`	ar ra y	<u>Check hook</u> execution data
`event.check.inter` `val`	in te g	The check execution frequency in seconds

		er	

`event.check.issue` `d`	in te g er	Time that the check request was issued in seconds since the Unix epoch
`event.check.label` `s`	m a p	Custom [labels](#) applied to the check
`event.check.last_` `ok`	in te g er	The last time that the check returned an OK status (`0`) in seconds since the Unix epoch
`event.check.low_fl` `ap_threshold`	in te g er	The check's flap detection low threshold in percent state change
`event.check.max_o` `utput_size`	in te g er	Maximum size of stored check outputs in bytes
`event.check.name`	st ri n g	Check name
`event.check.occur` `rences`	in te g er	The [number of preceding events](#) with the same status as the current event
`event.check.occur` `rences_watermark`	in te g er	For resolution events, the [number of preceding events](#) with a non-OK status
`event.check.outpu` `t`	st ri n	The output from the execution of the check command

		g	
`event.check.output_metric_format`	string	The [metric format](#) generated by the check command: `nagios_perfdata`, `graphite_plaintext`, `influxdb_line`, or `opentsdb_line`
`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_robin`	Boolean	Whether the check is configured to be executed in a [round-robin style](#)
`event.check.runtime_assets`	array	Sensu [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`	int	Exit status code produced by the check: `0` (OK), `1` (warning),

`s`	te g er	`2` (critical), or other status (unknown or custom status)	
`event.check.stdi n`	B o ol e a n	Whether the Sensu agent writes JSON-serialized entity and check data to the command process' STDIN	
`event.check.subsc riptions`	ar ra y	Subscriptions that the check belongs to	
`event.check.timeo ut`	in te g er	The check execution duration timeout in seconds	
`event.check.total _state_change`	in te g er	The total state change percentage for the check's history	
`event.check.ttl`	in te g er	The time-to-live (TTL) until the event is considered stale, in seconds	
`event.metrics.han dlers`	ar ra y	Sensu metric handlers assigned to the check	
`event.metrics.poi nts`	ar ra y	Metric data points including a name, timestamp, value, and tags	

Entity attributes available to filters

attribute	typ e	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.syst em.platform_famil y	strin g	The entity's operating system family
event.entity.syst em.platform_versio n	strin g	The entity's operating system version
event.entity.use r	strin g	Sensu RBAC username used by the agent entity

Event filter specification

Top-level attributes

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	`"type": "EventFilter"`

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	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the event filter, including the `name` and `namespace` 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. See 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 | ```
"metadata": {
  "name": "filter-weekdays-only",
  "namespace": "default",
  "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	
| | ```json
"spec": {
  "action": "allow",
  "expressions": [
    "event.entity.namespace == 'production'"
  ],
  "runtime_assets": []
}
``` |

Metadata attributes

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
| example | ```json
"name": "filter-weekdays-only"
``` |

namespace	
description	Sensu RBAC namespace that the event filter belongs to.
required	false
type	String
default	`default`
| example | ```json
"namespace": "production"
``` |

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	```"labels": {` ` "environment": "development",` ` "region": "us-west-2"` `}```

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 dashboard views.
required	false

type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

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

Spec attributes

action	
description	Action to take with the event if the event filter expressions match. See Inclusive and exclusive event filters for more information.
required	true
type	String
allowed values	`allow` , `deny`
example	``` "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`).
required	true
type	Array
example	``` "expressions": [```

```
          "event.check.team == 'ops'"
  ]
```

runtime_assets

description	Assets to apply to the event filter's execution context. JavaScript files in the lib directory of the asset will be evaluated.
required	false
type	Array of string
default	[]
example	`"runtime_assets": ["underscore"]`

Event filter examples

Minimum required filter attributes

YML

```
type: EventFilter
api_version: core/v2
metadata:
  name: filter_minimum
  namespace: default
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1
```

JSON

```
{
  "type": "EventFilter",
```

```json
    "api_version": "core/v2",
    "metadata": {
      "name": "filter_minimum",
      "namespace": "default"
    },
    "spec": {
      "action": "allow",
      "expressions": [
        "event.check.occurrences == 1"
      ]
    }
  }
```

Handle production events

The following event filter allows handling for only events with a custom entity label `"environment": "production"`:

YML

```yaml
type: EventFilter
api_version: core/v2
metadata:
  name: production_filter
  namespace: default
spec:
  action: allow
  expressions:
  - event.entity.labels['environment'] == 'production'
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "production_filter",
    "namespace": "default"
  },
  "spec": {
```

```
      "action": "allow",
      "expressions": [
        "event.entity.labels['environment'] == 'production'"
      ]
    }
  }
```

Handle 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: `action` 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
  namespace: default
spec:
  action: deny
  expressions:
  - event.entity.labels['environment'] == 'production'
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "not_production",
    "namespace": "default"
  },
  "spec": {
    "action": "deny",
    "expressions": [
```

```
      "event.entity.labels['environment'] == 'production'"
    ]
  }
}
```

Handle 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:
  annotations: null
  labels: null
  name: state_change_only
  namespace: default
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1
  runtime_assets: []
```

JSON

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "state_change_only",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
```

```
      ],
      "runtime_assets": []
    }
  }
```

Handle 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:
  annotations: null
  labels: null
  name: filter_interval_60_hourly
  namespace: default
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",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "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:
  annotations: null
  labels: null
  name: filter_interval_30_hourly
  namespace: default
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",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
```

```
      "action": "allow",
      "expressions": [
        "event.check.interval == 30",
        "event.check.occurrences == 1 || event.check.occurrences % 120 == 0"
      ],
      "runtime_assets": []
    }
  }
```

Handle 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:
  annotations: null
  labels: null
  name: nine_to_fiver
  namespace: default
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",
    "namespace": "default",
    "labels": null,
```

```
    "annotations": null
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5",
      "hour(event.timestamp) >= 9 && hour(event.timestamp) <= 17"
    ],
    "runtime_assets": []
  }
}
```

Use JavaScript libraries with Sensu filters

You can include JavaScript libraries in their event filter execution context with 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:
  annotations: null
  labels: null
  name: deny_if_failure_in_history
  namespace: default
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",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "spec": {
      "action": "deny",
      "expressions": [
        "_.reduce(event.check.history, function(memo, h) { return (memo || h.status !=
0); })"
      ],
      "runtime_assets": ["underscore"]
    }
}
```

Handlers

- ¬ [Pipe handlers](#)
- ¬ [TCP/UDP handlers](#)
- ¬ [Handler sets](#)
- ¬ [Keepalive event handlers](#)
- ¬ [Handler specification](#)
    - ¬ [Top-level attributes](#) | [Metadata attributes](#) | [Spec attributes](#) | `socket` [attributes](#)
- ¬ [Examples](#)

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 event data into arbitrary commands via `STDIN`
- ¬ **TCP/UDP handlers** send event data 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")

Discover, download, and share Sensu handlers assets using [Bonsai](#), the Sensu asset index. Read [Install plugins with assets](#) to get started.

Pipe handlers

Pipe handlers are external commands that can consume [event](#) data via STDIN.

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.

Handler sets

Handler set definitions allow you to use a single named handler set to refer to groups of handlers (individual collections of actions to take on event data).

NOTE: Attributes defined on 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.

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

```yml
type: Handler
api_version: core/v2
metadata:
  name: keepalive
  namespace: default
spec:
  handlers:
```

```
    - slack
  type: set
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "keepalive",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack"
    ]
  }
}
```

Handler specification

Top-level attributes

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	`"type": "Handler"`

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	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the handler that includes the `name` and `namespace` 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. See 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	`"metadata": {`
	`  "name": "handler-slack",`
	`  "namespace": "default",`
	`  "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 | ```json
"spec": {
  "type": "tcp",
  "socket": {
    "host": "10.0.1.99",
    "port": 4444
  },
  "metadata": {
    "name": "tcp_handler",
    "namespace": "default"
  }
}
``` |

Metadata attributes

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 | ```json
"name": "handler-slack"
``` |

namespace

description	Sensu RBAC namespace that the handler belongs to.
required	false
type	String
default	`default`
example	```"namespace": "production"```

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	```"labels": {```
	``` "environment": "development",```
	``` "region": "us-west-2"```
	```}```

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example	<pre>"annotations": {
	 "managed-by": "ops",
	 "playbook": "www.example.url"
	}</pre>

Spec attributes

type

description	Handler type.
required	true
type	String
allowed values	`pipe` , `tcp` , `udp` & `set`
example	<pre>"type": "pipe"</pre>

filters

description	Array of Sensu event filters (by names) to use when filtering events for the handler. Each array item must be a string.
required	false
type	Array
example	`"filters": ["occurrences", "production"]`

mutator

description	Name of the Sensu event mutator to use to mutate event data for the handler.
required	false
type	String
example	`"mutator": "only_check_output"`

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	`"timeout": 30`

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 (i.e. handlers configured with `"type": "pipe"`).
required	true (if `type` equals `pipe`)
type	String
example	`"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 (i.e. handlers configured with `"type": "pipe"`).
required	false
type	Array
example	`"env_vars":` `["API_KEY=0428d6b8nb51an4d95nbe28nf90865a66af5"]`

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 (i.e. handlers configured with `"type": "tcp"` or `"type": "udp"`).
required	true (if `type` equals `tcp` or `udp`)
type	Hash

example	``` "socket": {} ```

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 (i.e. handlers configured with* `"type": "set"` *).*
required	true (if `type` equals `set`)
type	Array
example	``` "handlers": ["pagerduty", "email", "ec2"] ```

runtime_assets

description	Array of <u>Sensu assets</u> (by names) required at runtime to execute the `command`
required	false
type	Array
example	``` "runtime_assets": ["ruby-2.5.0"] ```

`socket` attributes

host

description	Socket host address (IP or hostname) to connect to.

required	true
type	String
example	```"host": "8.8.8.8"```

port	
description	Socket port to connect to.
required	true
type	Integer
example	```"port": 4242```

Handler examples

Minimum required pipe handler attributes

YML

```yml
type: Handler
api_version: core/v2
metadata:
  name: pipe_handler_minimum
  namespace: default
spec:
  command: command-example
  type: pipe
```

JSON

```json
{
  "type": "Handler",
```

```
    "api_version": "core/v2",
    "metadata": {
      "name": "pipe_handler_minimum",
      "namespace": "default"
    },
    "spec": {
      "command": "command-example",
      "type": "pipe"
    }
  }
```

Minimum required TCP/UDP handler attributes

This example demonstrates a `tcp` type handler.Change the type from `tcp` to `udp` to create the minimum configuration for a `udp` type handler.

YML

```
type: Handler
api_version: core/v2
metadata:
  name: tcp_udp_handler_minimum
  namespace: default
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: tcp
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "tcp_udp_handler_minimum",
    "namespace": "default"
  },
  "spec": {
    "type": "tcp",
```

```
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Send Slack alerts

This handler will send alerts to a channel named `monitoring` with the configured webhook URL, using the `handler-slack` executable command.

YML

```
type: Handler
api_version: core/v2
metadata:
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  -
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXX
  filters:
  - is_incident
  - not_silenced
  handlers: []
  runtime_assets: []
  timeout: 0
  type: pipe
```

JSON

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

```json
      "namespace": "default"
    },
    "spec": {
      "command": "sensu-slack-handler --channel '#monitoring'",
      "env_vars": [

  "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXX
  XXXXXXXXXXXX"
      ],
      "filters": [
        "is_incident",
        "not_silenced"
      ],
      "handlers": [],
      "runtime_assets": [],
      "timeout": 0,
      "type": "pipe"
    }
  }
```

Send event data to a TCP socket

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

```yaml
type: Handler
api_version: core/v2
metadata:
  name: tcp_handler
  namespace: default
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: tcp
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "tcp_handler",
    "namespace": "default"
  },
  "spec": {
    "type": "tcp",
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Send event data to a UDP socket

This handler will forward event data to a UDP socket (10.0.1.99:4444) and timeout if an acknowledgement (`ACK`) is not received within 30 seconds.

YML

```yaml
type: Handler
api_version: core/v2
metadata:
  name: udp_handler
  namespace: default
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: udp
```

JSON

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

```
      "name": "udp_handler",
      "namespace": "default"
    },
    "spec": {
      "type": "udp",
      "socket": {
        "host": "10.0.1.99",
        "port": 4444
      }
    }
  }
```

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.

You can use registration events to execute one-time handlers for new Sensu agents to update an external configuration management database (CMDB). This example demonstrates how to configure a registration event handler to create or update a ServiceNow incident or event with the Sensu Go ServiceNow Handler:

YML

```
type: Handler
api_version: core/v2
metadata:
  name: registration
  namespace: default
spec:
  handlers:
  - servicenow-cmdb
  type: set
```

JSON

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

```
    "metadata": {
      "name": "registration",
      "namespace": "default"
    },
    "spec": {
      "handlers": [
        "servicenow-cmdb"
      ],
      "type": "set"
    }
  }
```

The agent reference describes agent registration and registration events in more detail.

Execute multiple handlers

The following example handler will execute three handlers: `slack` , `tcp_handler` , and `udp_handler` .

YML

```
type: Handler
api_version: core/v2
metadata:
  name: notify_all_the_things
  namespace: default
spec:
  handlers:
  - slack
  - tcp_handler
  - udp_handler
  type: set
```

JSON

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

```json
      "name": "notify_all_the_things",
      "namespace": "default"
    },
    "spec": {
      "type": "set",
      "handlers": [
        "slack",
        "tcp_handler",
        "udp_handler"
      ]
    }
  }
```

Health

- ¬ [Health payload example](#)

- ¬ [Health specification](#)

    - ¬ [Top-level attributes](#) | [ClusterHealth attributes](#) | [Header attributes](#)

Use Sensu's [health API](#) to make sure your backend is up and running and check the health of your etcd cluster members.

Health payload example

A request to the health endpoint retrieves a JSON map with health data for your Sensu instance.

```
curl -X GET \
http://127.0.0.1:8080/health

HTTP/1.1 200 OK
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 2882886652148554927,
      "MemberIDHex": "8923110df66458af",
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 4255616344056076734,
    "member_id": 2882886652148554927,
    "raft_term": 26
  }
}
```

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 | ```
"ClusterHealth": [
    {
        "MemberID": 2882886652148554927,
        "MemberIDHex": "8923110df66458af",
        "Name": "default",
        "Err": "",
        "Healthy": true
    }
]
``` |

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
  }
``` |

ClusterHealth attributes

Member ID	
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"`

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
``` |

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
```

Hooks

Hooks are reusable commands the agent executes in response to a check result before creating a monitoring event.You can create, manage, and reuse hooks independently of checks.Hooks enrich monitoring 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` .

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.See 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:

```
sensuctl event info entity_name check_name --format yaml
```

```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
```

Hook specification

Top-level attributes

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 | ```
"type": "HookConfig"
``` |

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 | ```
"api_version": "core/v2"
``` |

metadata

description	Top-level collection of metadata about the hook that includes the `name` and `namespace` 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. See 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 | ```
"metadata": {
  "name": "process_tree",
``` |

```
    "namespace": "default",
    "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	```"spec": {
	  "command": "ps aux",
	  "timeout": 60,
	  "stdin": false
	}```

Metadata attributes

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	
	`"name": "process_tree"`

namespace

description	The Sensu RBAC namespace that this hook belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	`"labels": {`

```
      "environment": "development",
      "region": "us-west-2"
    }
```

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example	<pre> "annotations": {
	   "managed-by": "ops",
	   "playbook": "www.example.url"
	  }</pre>

Spec attributes

command

description	Hook command to be executed.
required	true
type	String
example	<pre> "command": "sudo /etc/init.d/nginx start"</pre>

timeout

description	Hook execution duration timeout (hard stop). In seconds.
required	false
type	Integer
default	60
example	`"timeout": 30`

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	`"stdin": true`

runtime_assets

description	Array of Sensu assets (by their names) required at runtime for execution of the `command`.

required	false
type	Array
example	`"runtime_assets": ["log-context"]`

Examples

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

```yml
type: HookConfig
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: restart_nginx
  namespace: default
spec:
  command: sudo systemctl start nginx
  stdin: false
  timeout: 60
```

JSON

```json
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "restart_nginx",
```

```
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "spec": {
      "command": "sudo systemctl start nginx",
      "timeout": 60,
      "stdin": false
    }
  }
```

Capture the process tree

You can use hooks to automate data gathering for incident triage, For example, you can use a check hook to capture the process tree when a process is not running.

YML

```
type: HookConfig
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: process_tree
  namespace: default
spec:
  command: ps aux
  stdin: false
  timeout: 60
  runtime_assets: null
```

JSON

```
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "process_tree",
    "namespace": "default",
    "labels": null,
```

```
    "annotations": null
  },
  "spec": {
    "command": "ps aux",
    "timeout": 60,
    "stdin": false,
    "runtime_assets": null
  }
}
```

Check hook using 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:
  annotations: null
  labels:
    foo: bar
  name: tokensub
  namespace: default
spec:
  command: tokensub {{ .labels.foo }}
  stdin: false
  timeout: 60
```

JSON

```
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "annotations": null,
    "labels": {
```

```json
            "foo": "bar"
        },
        "name": "tokensub",
        "namespace": "default"
    },
    "spec": {
        "command": "tokensub {{ .labels.foo }}",
        "stdin": false,
        "timeout": 60
    }
}
```

Mutators

Handlers can specify a mutator to execute and transform event data before any handlers are applied.

- ¬ When the Sensu backend processes an event, it checks the handler for the presence of a mutator and executes that mutator before executing the handler.

- ¬ If the mutator executes successfully (returns an exit status code of `0`), the modified event data return to the handler and the handler is executed.

- ¬ 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.

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" ).
```

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 handler configuration `mutator` string:

YML

```yml
type: Handler
api_version: core/v2
metadata:
  name: graphite
  namespace: default
spec:
  mutator: only_check_output
  socket:
    host: 10.0.1.99
    port: 2003
  type: tcp
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "graphite",
    "namespace": "default"
  },
```

```
    "spec": {
      "type": "tcp",
      "socket": {
        "host": "10.0.1.99",
        "port": 2003
      },
      "mutator": "only_check_output"
    }
  }
```

Mutator specification

Mutators:

- ¬ Accept input/data via `STDIN`

- ¬ Can parse JSON event data

- ¬ Output JSON data (modified event data) to `STDOUT` or `STDERR`

- ¬ Produce an exit status code to indicate state:

    - ¬ `0` indicates OK status

    - ¬ exit codes other than `0` indicate failure

Top-level attributes

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	`"type": "Mutator"`

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	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the mutator that includes the `name` and `namespace` 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. See 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 | ```json
"metadata": {
  "name": "example-mutator",
  "namespace": "default",
  "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 | ```json
"spec": {
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
``` |

Metadata attributes

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 | ```json
"name": "example-mutator"
``` |

namespace	

description	Sensu RBAC namespace that the mutator belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	```"labels": {
"environment": "development",
"region": "us-west-2"
}```

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

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

Spec attributes

command	
description	Mutator command to be executed by the Sensu backend.
required	true
type	String
example	``` "command": "/etc/sensu/plugins/mutated.go" ```

env_vars	
description	Array of environment variables to use with command execution.
required	false

type	Array
example	`"env_vars": ["RUBY_VERSION=2.5.0"]`

timeout	
description	Mutator execution duration timeout (hard stop). In seconds.
required	false
type	integer
example	`"timeout": 30`

runtime_assets	
description	Array of Sensu assets (by their names) required at runtime for execution of the `command`.
required	false
type	Array
example	`"runtime_assets": ["ruby-2.5.0"]`

Examples

Example mutator definition

The following Sensu mutator definition uses an imaginary Sensu plugin, `example_mutator.go`, to

modify event data prior to handling the event.

YML

```yaml
type: Mutator
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: example-mutator
  namespace: default
spec:
  command: example_mutator.go
  env_vars: []
  runtime_assets: []
  timeout: 0
```

JSON

```json
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
  }
}
```

Minimum required mutator attributes

YML

```yaml
type: Mutator
```

```
api_version: core/v2
metadata:
  name: mutator_minimum
  namespace: default
spec:
  command: example_mutator.go
```

JSON

```json
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "mutator_minimum",
    "namespace": "default"
  },
  "spec": {
    "command": "example_mutator.go"
  }
}
```

Role-based access control

Sensu role-based access control (RBAC) helps different teams and projects share a Sensu instance. RBAC allows you to manage user access and resources based on namespaces, groups, roles, and bindings.

- **Namespaces** partition resources within Sensu. Sensu entities, checks, handlers, and other namespaced resources belong to a single namespace.

- **Roles** create sets of permissions (e.g. 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.

Sensu access controls apply to sensuctl, the Sensu API, and the Sensu dashboard. In addition to built-in RBAC, Sensu includes commercial support for authentication using external authentication providers.

Namespaces

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.

To create and manage namespaces, configure sensuctl as the default `admin` user or create a cluster role with `namespaces` permissions.

Default namespaces

Every Sensu backend includes a `default` namespace. All resources created without a specified namespace are created within the `default` namespace.

Manage namespaces

You can use sensuctl to view, create, and delete namespaces. To get help with managing namespaces with sensuctl:

```
sensuctl namespace help
```

View namespaces

You can use sensuctl to view all namespaces within Sensu:

```
sensuctl namespace list
```

NOTE: *For users on supported Sensu Go distributions,* `sensuctl namespace list` *lists only the namespaces that the current user has access to.*

Create namespaces

You can use sensuctl to create a namespace. For example, the following command creates a namespace called `production`:

```
sensuctl namespace create production
```

Namespace names can contain alphanumeric characters and hyphens and must begin and end with an alphanumeric character.

Delete namespaces

To delete a namespace:

```
sensuctl namespace delete [NAMESPACE-NAME]
```

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

```yaml
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
  }
}
```

See the <u>reference docs</u> for the corresponding <u>resource type</u> 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. See the <u>sensuctl reference</u> for more information.

Namespace specification

Attributes

name	
description	Name of the namespace. Names can contain alphanumeric characters and hyphens and must begin and end with an alphanumeric character.
required	true
type	String
example	`"name": "production"`

Namespace example

This example is in `yml` and `wrapped-json` formats for use with `sensuctl create` :

YML

```yml
type: Namespace
api_version: core/v2
metadata: {}
spec:
  name: default
```

JSON

```json
{
  "type": "Namespace",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "name": "default"
  }
}
```

Resources

Permissions within Sensu are scoped to resource types, like checks, handlers, and users. You can use resource types to configure permissions in Sensu roles and cluster roles.

Namespaced resource types

Namespaced resources must belong to a single namespace. You can access namespaced resources by roles and cluster roles.

type	description
`assets`	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
`rolebindings`	Namespace-specific role assigners
`roles`	Namespace-specific permission sets
`silenced`	Silencing resources within a namespace

Cluster-wide resource types

Cluster-wide resources cannot be assigned to a namespace. You can access cluster-wide resources

only by cluster roles.

type	description
`authproviders`	Authentication provider configuration (commercial feature)
`cluster`	Sensu clusters running multiple Sensu backends
`clusterrolebindings`	Cluster-wide role assigners
`clusterroles`	Cluster-wide permission sets
`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
`users`	People or agents that interact with Sensu

Special resource types

You can access special resource types by both 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 and groups to

one or more roles.Users and groups inherit all permissions from each role assigned to them.

Use your Sensu username and password to <u>configure sensuctl</u> or log in to the <u>dashboard</u>.

Default users

When you install the Sensu backend, during the <u>initialization step</u>, you create a username and password for a `default` namespace.

NOTE: If you are using Docker, the `sensu-backend init` command runs automatically with a default username (`admin`) and password (`P@ssw0rd!`). You do not need to create a username and password for the `default` namespace if you are using Docker.

This is the user that you can use to manage all aspects of Sensu and create new users.

attribute	value
username	`YOUR_USERNAME`
password	`YOUR_PASSWORD`
groups	`cluster-admins`
cluster role	`cluster-admin`
cluster role binding	`cluster-admin`

Once authenticated, you can change the default password for the admin user with the `change-password` command:

```
sensuctl user change-password
```

Sensu also includes an `agent` user, which is used internally by the Sensu agent.You can configure `agent` user credentials with the `user` and `password` <u>agent configuration flags</u>.

Manage users

To test the password for a user created with Sensu's built-in <u>basic authentication</u>:

```
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 provider. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP) or Active Directory (AD).

To change the password for a user:

```
sensuctl user change-password USERNAME --current-password CURRENT_PASSWORD --new-password NEW_PASSWORD
```

To disable a user:

```
sensuctl user disable USERNAME
```

To re-enable a disabled user:

```
sensuctl user reinstate USERNAME
```

View users

You can use sensuctl to see a list of all users within Sensu.

To return a list of users in `yaml` format for use with `sensuctl create`:

```
sensuctl user list --format yaml
```

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.

Passwords must have at least eight characters.

```
sensuctl user create alice --password='password' --groups=ops,dev
```

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.

User specification

Attributes

username	
description	Name of the user. Cannot contain special characters.
required	true
type	String
example	```"username": "alice"```

password	
description	User's password. Passwords must have at least eight characters.

required	true
type	String
example	`"password": "USER_PASSWORD"`

groups

description	Groups to which the user belongs.
required	false
type	Array
example	`"groups": ["dev", "ops"]`

disabled

description	If `true`, the user's account is disabled. Otherwise, `false`.
required	false
type	Boolean
default	`false`
example	`"disabled": false`

User example

The following example is in `yml` and `wrapped-json` formats for use with `sensuctl create`.

YML

```
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  groups:
  - ops
  - dev
  password: USER_PASSWORD
  username: alice
```

JSON

```json
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "username": "alice",
    "password": "USER_PASSWORD",
    "disabled": false,
    "groups": ["ops", "dev"]
  }
}
```

Groups

A group is a set of users within Sensu. You can assign groups to one or more roles, and you can assign users to one or more groups. Groups inherit all permissions from each role assigned to them.

Groups are not a resource type within Sensu. You can create and manage groups 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.

Manage groups

Assign a user to a group

Groups are created and managed within user definitions. You can use sensuctl to add users to groups.

To add a user to a group:

```
sensuctl user add-group USERNAME GROUP
```

To set the groups for a user:

```
sensuctl user set-groups USERNAME GROUP1[,GROUP2, ...[,GROUPN]]
```

Remove a user from a group

You can use sensuctl to remove users from groups.

To remove a user from a group:

```
sensuctl user remove-group USERNAME GROUP
```

To remove a user from all groups:

```
sensuctl user remove-groups USERNAME
```

Roles and cluster roles

A role is a set of permissions that control access to Sensu resources. Roles specify permissions for

resources within a namespace.Cluster role can include permissions for cluster-wide resources.

You can use role bindings to assign roles to user and groups.To avoid recreating commonly used roles in each namespace, create a cluster role and use a role binding (not a cluster role binding) to restrict permissions within a specific namespace.

To create and manage roles cluster-wide, configure sensuctl as the default `admin` user or create a cluster role with `roles` permissions.To create and manage roles within a namespace, create a role with `roles` permissions within that namespace.

Cluster roles

Cluster roles can specify access permissions for cluster-wide resources like users and namespaces as well as namespaced resources like checks and handlers.They can also be used to grant access to namespaced resources across all namespaces (for example, to run `sensuctl check list --all-namespaces`) when used in conjunction with cluster role bindings.

Cluster roles use the same specification as roles and can be managed using the same sensuctl commands with `cluster-role` substituted for `role`.

To create and manage cluster roles, configure sensuctl as the default `admin` user or create a cluster role with permissions for `clusterroles`.

Default roles

Every Sensu backend includes:

role name	type	description
`cluster-admin`	Cluster Role	Full access to all resource types across namespaces, including access to cluster-wide resource types.
`admin`	C	Full access to all resource types. You can apply this cluster role

	lu st er Ro l e	within a namespace by using a role binding (not a cluster role binding).
`edit`	C lu st er Ro l e	Read and write access to most resources except roles and role bindings. You can apply this cluster role within a namespace by using a role binding (not a cluster role binding).
`view`	C lu st er Ro l e	Read-only permission to most resource types with the exception of roles and role bindings. You can apply this cluster role within a namespace by using a role binding (not a cluster role binding).
`system:agent`	C lu st er Ro l e	Used internally by Sensu agents. You can configure an agent's user credentials using the `user` and `password` agent configuration flags.

Manage roles and cluster roles

You can use sensuctl to view, create, edit, and delete roles and cluster roles.

NOTE: To use any of these example commands with cluster roles, substitute the `cluster-role` *command for the* `role` *command.*

To get help managing roles with sensuctl:

```
sensuctl role help
```

To edit a role:

```
sensuctl edit role [ROLE-NAME] [flags]
```

View roles and cluster roles

You can use sensuctl to see a list of roles within Sensu:

```
sensuctl role list
```

To see the permissions and scope for a specific role:

```
sensuctl role info admin
```

To view cluster roles, use the `cluster-role` command:

```
sensuctl cluster-role list
```

Create roles

You can use sensuctl to create a role.For example, the following command creates an admin role restricted to the production namespace.

```
sensuctl role create prod-admin --verb get,list,create,update,delete --resource * --
namespace production
```

After you create a role, create a role binding (or cluster role binding) to assign the role to users and groups.For example, to assign the `prod-admin` role created above to the `oncall` group, create this role binding:

```
sensuctl role-binding create prod-admin-oncall --role=prod-admin --group=oncall
```

Create cluster-wide roles

You can use sensuctl to create a cluster role. For example, the following command creates a global event reader role that can read only events across all namespaces within Sensu.

```
sensuctl cluster-role create global-event-reader --verb get,list --resource events
```

Delete roles and cluster roles

To delete a role:

```
sensuctl role delete [ROLE-NAME]
```

Role and cluster role specification

Role and cluster role attributes

name	
description	Name of the role.
required	true
type	String
example	`"name": "admin"`

namespace	
description	Namespace the role is restricted to. This attribute is not available for cluster roles.
required	false
type	String
example	`"namespace": "production"`

rules	
description	Rulesets that the role applies.
required	true
type	Array

example	```"rules": [
	  {
	    "verbs": ["get", "list"],
	    "resources": ["checks"],
	    "resource_names": [""]
	  }
	]```

Rule attributes

A rule is an explicit statement that grants a particular permission to a resource.

verbs	
description	Permissions to be applied by the rule: `get`, `list`, `create`, `update`, or `delete`.

required	true
type	Array
example	`"verbs": ["get", "list"]`

resources

description	Type of resource that the rule has permission to access. Roles can only access namespaced resource types. Cluster roles can access namespaced and cluster-wide resource types. See resource types for available types.
required	true
type	Array
example	`"resources": ["checks"]`

resource_names

description	Specific resource names that the rule has permission to access. Resource name permissions are only taken into account for requests using `get`, `update`, and `delete` verbs.
required	false
type	Array
example	`"resource_names": ["check-cpu"]`

Role and cluster role examples

These examples are in `yml` and `wrapped-json` formats for use with `sensuctl create`.

Role example

YML

```
type: Role
api_version: core/v2
metadata:
  name: namespaced-resources-all-verbs
  namespace: default
spec:
  rules:
  - resource_names: []
    resources:
    - assets
    - checks
    - entities
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - rolebindings
    - roles
    - silenced
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "namespaced-resources-all-verbs",
    "namespace": "default"
  },
```

```
    "spec": {
      "rules": [
        {
          "resource_names": [],
          "resources": [
            "assets", "checks", "entities", "events", "filters", "handlers",
            "hooks", "mutators", "rolebindings", "roles", "silenced"
          ],
          "verbs": ["get", "list", "create", "update", "delete"]
        }
      ]
    }
  }
```

Cluster role example

YML

```
type: ClusterRole
api_version: core/v2
metadata:
  name: all-resources-all-verbs
spec:
  rules:
  - resource_names: []
    resources:
    - assets
    - checks
    - entities
    - events
    - filters
    - handlers
    - hooks
    - mutators
    - rolebindings
    - roles
    - silenced
    - cluster
    - clusterrolebindings
    - clusterroles
    - namespaces
    - users
```

```
        - authproviders
        - license
      verbs:
        - get
        - list
        - create
        - update
        - delete
```

JSON

```
  {
    "type": "ClusterRole",
    "api_version": "core/v2",
    "metadata": {
      "name": "all-resources-all-verbs"
    },
    "spec": {
      "rules": [
        {
          "resource_names": [],
          "resources": [
            "assets", "checks", "entities", "events", "filters", "handlers",
            "hooks", "mutators", "rolebindings", "roles", "silenced",
            "cluster", "clusterrolebindings", "clusterroles",
            "namespaces", "users", "authproviders", "license"
          ],
          "verbs": ["get", "list", "create", "update", "delete"]
        }
      ]
    }
  }
```

Role bindings and cluster role bindings

A role binding assigns a *role* or *cluster role* to users and groups within a namespace. A cluster role binding assigns a *cluster role* to users and groups across namespaces and resource types.

Cluster role bindings use the same specification as role bindings and can be managed using the same

sensuctl commands with `cluster-role-binding` substituted for `role-binding` .

To create and manage role bindings within a namespace, create a role with `rolebindings` permissions within that namespace, and log in by configuring sensuctl.

To create and manage cluster role bindings, configure sensuctl as the default `admin` user or create a cluster role with permissions for `clusterrolebindings` .

Manage role bindings and cluster role bindings

You can use sensuctl to view, create, and delete role bindings and cluster role bindings.

NOTE: To use any of these commands with cluster roles, substitute the `cluster-role-binding` command for the `role-binding` command.

To get help managing role bindings with sensuctl:

```
sensuctl role-binding help
```

View role bindings and cluster role bindings

You can use sensuctl to see a list of role bindings within Sensu:

```
sensuctl role-binding list
```

To see the details for a specific role binding:

```
sensuctl role-binding info [BINDING-NAME]
```

To see a list of cluster role bindings:

```
sensuctl cluster-role-binding list
```

Create role bindings and cluster role bindings

You can use sensuctl to see a create a role binding that assigns a role:

```
sensuctl role-binding create [NAME] --role=NAME [--user=username] [--
group=groupname]
```

To create a role binding that assigns a cluster role:

```
sensuctl role-binding create [NAME] --cluster-role=NAME [--user=username] [--
group=groupname]
```

To create a cluster role binding:

```
sensuctl cluster-role-binding create [NAME] --cluster-role=NAME [--user=username] [-
-group=groupname]
```

Delete role bindings and cluster role bindings

To delete a role binding:

```
sensuctl role-binding delete [ROLE-NAME]
```

Role binding and cluster role binding specification

roleRef	
description	Reference a role in the current namespace or a cluster role.
required	true
type	Hash

example	
	```json "roleRef": { "type": "Role", "name": "event-reader" } ```

subjects

description	Users or groups being assigned.
required	true
type	Array
example	```json "subjects": [{ "type": "User", "name": "alice" }] ```

`roleRef` *specification*

type

description	`Role` for a role binding or `ClusterRole` for a cluster role binding.
required	true
type	String
example	```json "type": "Role" ```

name	
description	Name of the role or cluster role being assigned.
required	true
type	String
example	`"name": "event-reader"`

`subjects` *specification*

type	
description	`User` for assigning a user or `Group` for assigning a group.
required	true
type	String
example	`"type": "User"`

name	
description	Username or group name.
required	true
type	String
example	`"name": "alice"`
example with prefix	

```
          "name": "ad:alice"
```

Role binding and cluster role binding examples

These examples are in `yml` and `wrapped-json` formats for use with `sensuctl create` .

Role binding example
YML

```yaml
type: RoleBinding
api_version: core/v2
metadata:
  name: event-reader-binding
  namespace: default
spec:
  role_ref:
    name: event-reader
    type: Role
  subjects:
  - name: bob
    type: User
```

JSON

```json
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "event-reader-binding",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "event-reader",
      "type": "Role"
    },
    "subjects": [
      {
```

```json
      "name": "bob",
      "type": "User"
    }
  ]
  }
}
```

Cluster role binding example

YML

```yaml
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

```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"
      }
```

```
      ]
    }
  }
```

Role and role binding example

The following role and role binding give a `dev` group access to create and manage Sensu workflows within the `default` namespace.

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "workflow-creator",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
        "resource_names": [],
        "resources": ["checks", "hooks", "filters", "events", "filters", "mutators",
"handlers"],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "dev-binding",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "workflow-creator",
      "type": "Role"
    },
    "subjects": [
```

```
      {
        "name": "dev",
        "type": "Group"
      }
    ]
  }
}
```

Role and role binding example 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.

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "workflow-creator",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
        "resource_names": [],
        "resources": ["checks", "hooks", "filters", "events", "filters", "mutators",
"handlers"],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "dev-binding-with-groups-prefix",
    "namespace": "default"
  },
```

```
    "spec": {
      "role_ref": {
        "name": "workflow-creator",
        "type": "Role"
      },
      "subjects": [
        {
          "name": "ad:dev",
          "type": "Group"
        }
      ]
    }
  }
```

Example workflows

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`.

```
  {
    "type": "User",
    "api_version": "core/v2",
    "metadata": {},
    "spec": {
      "disabled": false,
      "username": "alice"
    }
  }
  {
```

```json
    "type": "Role",
    "api_version": "core/v2",
    "metadata": {
      "name": "default-admin",
      "namespace": "default"
    },
    "spec": {
      "rules": [
        {
          "resource_names": [],
          "resources": [
            "assets", "checks", "entities", "events", "filters", "handlers",
            "hooks", "mutators", "rolebindings", "roles", "silenced"
          ],
          "verbs": ["get", "list", "create", "update", "delete"]
        }
      ]
    }
  }
  {
    "type": "RoleBinding",
    "api_version": "core/v2",
    "metadata": {
      "name": "alice-default-admin",
      "namespace": "default"
    },
    "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` .

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice"
  }
}
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
        "resource_names": [],
        "resources": [
          "assets", "checks", "entities", "events", "filters", "handlers",
          "hooks", "mutators", "rolebindings", "roles", "silenced"
        ],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
```

```
    }
    {
      "type": "RoleBinding",
      "api_version": "core/v2",
      "metadata": {
        "name": "ops-default-admin",
        "namespace": "default"
      },
      "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`.

```
    {
```

```json
    "type": "User",
    "api_version": "core/v2",
    "metadata": {},
    "spec": {
      "disabled": false,
      "username": "alice",
      "groups": ["ops"]
    }
  }
  {
    "type": "ClusterRole",
    "api_version": "core/v2",
    "metadata": {
      "name": "default-admin"
    },
    "spec": {
      "rules": [
        {
          "resource_names": [],
          "resources": [
            "assets", "checks", "entities", "events", "filters", "handlers",
            "hooks", "mutators", "rolebindings", "roles", "silenced",
            "cluster", "clusterrolebindings", "clusterroles",
            "namespaces", "users", "authproviders", "license"
          ],
          "verbs": ["get", "list", "create", "update", "delete"]

        }
      ]
    }
  }
  {
    "type": "ClusterRoleBinding",
    "api_version": "core/v2",
    "metadata": {
      "name": "ops-default-admin"
    },
    "spec": {
      "role_ref": {
        "name": "default-admin",
        "type": "ClusterRole"
      },
```

```json
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

Sensu query expressions

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

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
```

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
```

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.indexOf('webserver') >= 0
```

Likewise, this expression returns `true` if the event's entity includes the annotation `www.company.com`:

```
event.entity.annotations.indexOf('www.company.com') >= 0
```

Silencing

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`. 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 the check name or subscription described in a silencing entry matches an event and the handler uses the `not_silenced` built-in filter, the handler will not be executed.

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. The presence of silences indicates that the event is silenced.

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.

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	`"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

example	
	```"api_version": "core/v2"```

metadata

description	Top-level collection of metadata about the silencing entry that includes the `name` and `namespace` 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. See 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
| example | ```
"metadata": {
  "name": "appserver:mysql_status",
  "namespace": "default",
  "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 | ```
"spec": {
  "expire": -1,
  "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	`"name": "appserver:mysql_status"`

namespace

description	Sensu RBAC namespace that the silencing entry belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data that you can use for response and dashboard view filtering.
	If you include labels in your event data, you can filter API responses, sensuctl responses, and dashboard 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	

```
"labels": {
  "environment": "development",
  "region": "us-west-2"
}
```

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 dashboard views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	

```
"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	`"check": "haproxy_status"`

subscription	
description	Name of the subscription the entry should match.
required	true, unless `check` is provided
type	String
example	`"subscription": "entity:i-424242"`

begin	
description	Time at which silence entry goes into effect. In epoch.
required	false
type	Integer

example	
	```"begin": 1512512023```

expire

description	Number of seconds until the entry should be deleted.
required	false
type	Integer
default	-1
example	```"expire": 3600```

expire_on_resolve

description	`true` if the entry should be deleted when a check begins to return OK status (resolves). Otherwise, `false`.
required	false
type	Boolean
default	false
example	```"expire_on_resolve": true```

creator

description	Person, application, or entity responsible for creating the entry.
required	false

type	String
default	null
example	

```
"creator": "Application Deploy Tool 5.0"
```

reason

description	Explanation of the reason for creating the entry.
required	false
type	String
default	null
example	

```
"reason": "rebooting the world"
```

Examples

Silence all checks on a specific entity

Suppose you want to silence any alerts on the Sensu entity `i-424242`. To do this, use per-entity subscriptions:

YML

```yml
type: Silenced
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: entity:i-424242:*
  namespace: default
spec:
```

```
begin: 1542671205
check: null
creator: admin
expire: -1
expire_on_resolve: false
reason: null
subscription: entity:i-424242
```

JSON

```json
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "entity:i-424242:*",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": null,
    "check": null,
    "subscription": "entity:i-424242",
    "begin": 1542671205
  }
}
```

Silence a specific check on a specific entity

To continue the previous example, here's how to silence a check named `check_ntp` on entity `i-424242`, ensuring the entry is deleted after the underlying issue is resolved:

YML

```yaml
check: check_ntp
expire_on_resolve: true
subscription: entity:i-424242
```

JSON

```json
{
  "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.

Silence all checks on entities with a specific subscription

In this example, you'll completely silence any entities subscribed to `appserver`. Just as in the example of silencing all checks on a specific entity, you'll create a silencing entry that specifies only the `appserver` subscription:

YML

```yml
subscription: appserver
```

JSON

```json
{
  "subscription": "appserver"
}
```

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
check: mysql_status
subscription: appserver
```

```json
{
  "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
check: mysql_status
```

```json
{
  "check": "mysql_status"
}
```

Delete a silence

To delete a silencing entry, you must provide its name.

Subscription-only silencing entry names will be similar to this example:

```yml
name: appserver:*
```

JSON

```json
{
  "name": "appserver:*"
}
```

Check-only silencing entry names will be similar to this example:

YML

```yml
name: '*:mysql_status'
```

JSON

```json
{
  "name": "*:mysql_status"
}
```

Tessen

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. See Troubleshooting to set the Sensu backend log level and view logs.

Configure Tessen

You can use the Tessen API and sensuctl to view and manage Tessen configuration. Tessen is enabled by default on Sensu backends and required for licensed Sensu instances. To manage Tessen configuration with sensuctl, configure sensuctl as the default `admin` user.

To see Tessen status:

```
sensuctl tessen info
```

To opt out of Tessen:

```
sensuctl tessen opt-out
```

NOTE: *Licensed Sensu instances override Tessen configuration 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

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	`"type": "TessenConfig"`

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	`"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	`"spec": {`
	`  "opt_out": false`
	`}`

Spec attributes

opt_out

description	`true` to opt out of Tessen. Otherwise, `false` . Licensed Sensu instances override the `opt_out` attribute to `false` at runtime.
required	true
type	Boolean
default	`false`
example	`opt_out": false`

Tessen configuration example

This example is in `wrapped-json` format for use with `sensuctl create` . To manage Tessen with the Tessen API , use non-wrapped `json` format as shown in the API docs.

YML

```yml
type: TessenConfig
api_version: core/v2
spec:
  opt_out: false
```

JSON

```json
{
  "type": "TessenConfig",
  "api_version": "core/v2",
  "spec": {
    "opt_out": false
  }
}
```

Tessen payload example

If you opt in to Tessen, Sensu sends various metrics back to the Tessen service. In the example payload below, Sensu is sending the number of check hooks back to the Tessen service.

```json
{
    "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:

```json
{
    "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:

```json
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "filter_count",
    "metric_value": 4,
    "msg": "collected a metric for tessen",
    "time": "2019-09-16T09:02:01Z"
}
```

If you opt into Tessen, you can view all of the metrics and payloads in the logs:

```
journalctl _COMM=sensu-backend.service
```

To view the events on-disk, see Log Sensu services with systemd.

Tokens

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 definition `command` attributes and hook `command` attributes. Only entity attributes are available for substitution. Available attributes will always have string values, such as labels and annotations.

Manage entity labels

You can use token substitution with any defined entity attributes, including custom labels. See the entity reference for information about managing entity labels for proxy entities and agent entities.

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 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 (e.g. `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.

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 tokens cannot be used as bare integer values or to access individual list items.

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*.

Examples

Token substitution for check thresholds

In this example hook and check configuration, the `check-disk-usage.go` command accepts `-w` (warning) and `-c` (critical) arguments to indicate the thresholds (as percentages) for creating warning or critical events. If no token substitutions are provided by an entity configuration, Sensu will use default values to create a warning event at 80% disk capacity (i.e. `{{ .labels.disk_warning | default 80 }}`) and a critical event at 90% capacity (i.e. `{{ .labels.disk_critical | default 90 }}`).

Hook configuration:

YML

```yaml
type: HookConfig
api_version: core/v2
metadata:
  name: disk_usage_details
  namespace: default
spec:
  command: du -h --max-depth=1 -c {{index .labels "disk_usage_root" | default "/"}} 2>/dev/null
  runtime_assets: null
  stdin: false
  timeout: 60
```

JSON

```json
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "disk_usage_details",
    "namespace": "default"
  },
  "spec": {
    "command": "du -h --max-depth=1 -c {{index .labels \"disk_usage_root\" | default \"/\"}}  2>/dev/null",
    "runtime_assets": null,
    "stdin": false,
    "timeout": 60
  }
}
```

Check configuration:

YML

```yaml
type: CheckConfig
api_version: core/v2
metadata:
  name: check-disk-usage
  namespace: default
spec:
  check_hooks:
  - non-zero:
    - disk_usage_details
  command: check-disk-usage.rb -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
  proxy_entity_name: ""
  publish: true
  round_robin: false
```

```
  runtime_assets: null
  stdin: false
  subdue: null
  subscriptions:
  - staging
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-disk-usage",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": [
      {
        "non-zero": [
          "disk_usage_details"
        ]
      }
    ],
    "command": "check-disk-usage.rb -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,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "staging"
```

```
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

The following example <u>entity</u> provides the necessary attributes to override the
`.labels.disk_warning` and `labels.disk_critical` tokens declared above:

YML

```
type: Entity
api_version: core/v2
metadata:
  annotations: null
  labels:
    disk_critical: "90"
    disk_warning: "80"
  name: example-hostname
  namespace: default
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:example-hostname
  - staging
  system:
    arch: amd64
    hostname: example-hostname
```

```yaml
      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
    user: agent
```

JSON

```json
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "example-hostname",
    "namespace": "default",
    "labels": {
      "disk_warning": "80",
      "disk_critical": "90"
    },
    "annotations": null
  },
  "spec": {
    "entity_class": "agent",
    "system": {
      "hostname": "example-hostname",
      "os": "linux",
```

```json
      "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"
    },
    "subscriptions": [
      "entity:example-hostname",
      "staging"
    ],
    "last_seen": 1542667231,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
```

```
            "passwd",
            "pass",
            "api_key",
            "api_token",
            "access_key",
            "secret_key",
            "private_key",
            "secret"
        ]
    }
}
```

License management

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.

Sensu Go License

View and download your Sensu Go license key.

Account ID

44

Billing Email

Issued

February 19, 2019

Expires

February 19, 2020

Download license

With the license file downloaded, you can activate your license with sensuctl or the license API.

To activate your license with sensuctl:

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

Use sensuctl to view your license details at any time.

```
# Active license
sensuctl license info
=== Training Team - Sensu
Account Name: Training Team - Sensu
Account ID:   123
Plan:         managed
Version:      1
Features:     all
EntityLimit:  0
Issuer:       Sensu, Inc.
Issued:       2019-02-15 15:01:44 -0500 -0500
Valid:        true
Valid Until:  2019-03-15 00:00:00 -0800 -0800

# No license found
sensuctl license info
Error: not found
```

Entity limit

Your commercial license includes the entity limit tied to your Sensu licensing package. An entity limit of `0` allows unlimited entities. Both agent and proxy entities count toward the overall entity limit. Contact Sensu to upgrade your commercial license.

To see your current entity count, use 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: Bearer $SENSU_ACCESS_TOKEN"
```

Your current entity count and limit are listed as response headers:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Entity-Count: 4
Sensu-Entity-Limit: 0
```

License expiration

To see your commercial license expiration date, log in to your Sensu account.

If your license is within 30 days of expiration, Sensu issues regular warnings in the Sensu backend logs. 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
- ¬ Configure authentication providers
- ¬ Use the license management API
- ¬ Discover enterprise assets
- ¬ Install plugins with assets
- ¬ Contact Sensu support
- ¬ Contact Sensu sales

Learn Sensu Go with interactive training

- ¬ [Learn Sensu in 15 minutes](#)

- ¬ [Up and running with Sensu Go](#)

- ¬ [Send Sensu Go alerts to PagerDuty](#)

Sensu is the industry-leading telemetry and service health-checking solution for multi-cloud monitoring at scale.

Our interactive training tutorials help you get started with Sensu Go, using only your browser. With these tutorials, you can learn how to automate your monitoring workflows, gain deep visibility into systems that are important to your business operations, get complete control over your alerts, and integrate anywhere, including with the tools you're already using.

Learn Sensu in 15 minutes

This interactive tutorial demonstrates how to:

- ¬ Deploy a basic Sensu stack.

- ¬ Log in to the Sensu web UI.

- ¬ Create a monitoring event and use Sensu to send alerts based on the event to a Slack channel.

- ¬ Use a Sensu agent to automatically produce events, then store event data with InfluxDB and visualize it with Grafana.

Launch **Learn Sensu in 15 minutes**.

Up and running with Sensu Go

This interactive tutorial will help you get Sensu Go up and running from scratch, using only your browser. We've provisioned a CentOS 7 host for you, with an Nginx webserver already installed and running. When you complete this tutorial, your system will have both the Sensu backend and agent running to monitor the local Nginx service.

Launch **Up and running with Sensu Go**.

Send Sensu Go alerts to PagerDuty

When you complete this interactive tutorial, your Sensu Go backend will be configured with a handler that will send critical alerts to your PagerDuty account. In this scenario, you will:

- ¬ Add a Sensu Nagios Foundation asset.

- ¬ Add the PagerDuty asset and create a handler that uses your PagerDuty API key.

- ¬ Send an alert for a Sensu Go event to PagerDuty.

Launch **Send Sensu Go alerts to PagerDuty**.

Learn Sensu Go

In this tutorial, you'll download the Sensu sandbox and create a monitoring workflow with Sensu.

Set up the sandbox

1. Install Vagrant and VirtualBox

- ⌐ Download Vagrant

- ⌐ Download VirtualBox

2. Download the sandbox

Download from GitHub or clone the repository:

```
git clone https://github.com/sensu/sandbox && cd sandbox/sensu-go
```

NOTE: *If you've cloned the sandbox repository before, run* `cd sandbox/sensu-go` *and* `git pull https://github.com/sensu/sandbox` *instead.*

3. Start Vagrant

```
ENABLE_SENSU_SANDBOX_PORT_FORWARDING=1 vagrant up
```

The Learn Sensu sandbox is a CentOS 7 virtual machine pre-installed with Sensu, InfluxDB, and

Grafana.It's intended for you to use as a learning tool — we do not recommend using it in a production installation.To install Sensu in production, use the underline installation guide instead.

The sandbox startup process takes about 5 minutes.

NOTE: The sandbox configures VirtualBox to forward TCP ports 3002 and 4002 from the sandbox virtual machine (VM) to the localhost to make it easier for you to interact with the sandbox dashboards. Dashboard links provided in this tutorial assume port forwarding from the VM to the host is active.

4. SSH into the sandbox

Thanks for waiting! To start, shell into the sandbox:

```
vagrant ssh
```

You should be greeted with this prompt:

```
[sensu_go_sandbox]$
```

- ⌐ To exit the sandbox, press `CTRL` + `D` .
- ⌐ To erase and restart the sandbox, run `vagrant destroy` and then `vagrant up` .
- ⌐ To reset the sandbox's Sensu configuration to the beginning of this tutorial, run `vagrant provision` .

NOTE: The sandbox pre-configures sensuctl with the Sensu Go admin user, so you won't have to configure sensuctl each time you spin up the sandbox to try out a new feature. Before installing sensuctl outside of the sandbox, read the first time setup reference to learn how to configure sensuctl.

Lesson #1: Create a Sensu monitoring event

First, make sure everything is working correctly using the sensuctl command line tool.Use sensuctl to see that your Sensu backend instance has a single namespace, `default` , and two users: the default admin user and the user created for a Sensu agent to use.

```
sensuctl namespace list
```

```
Name
_____

default
```

```
sensuctl user list
```

```
Username    Groups        Enabled
_____ _____ _____

admin    cluster-admins   true
agent    system:agents    true
```

Sensu keeps track of monitored components as entities.Start by using sensuctl to make sure Sensu hasn't connected to any entities yet:

```
sensuctl entity list
```

```
ID  Class  OS  Subscriptions  Last Seen
____ _____ ____ _____ _____
```

Now you can start the Sensu agent to begin monitoring the sandbox:

```
sudo systemctl start sensu-agent
```

Use sensuctl to see that Sensu is now monitoring the sandbox entity:

```
sensuctl entity list
```

```
ID      Class  OS      Subscriptions      Last Seen
```

```
_____ _____ _____ _____
_____
    sensu-go-sandbox   agent   linux   entity:sensu-go-sandbox   2019-01-24 21:29:06 +0000 UTC
```

Sensu agents send keepalive events to help you monitor agent status. Use sensuctl to see the keepalive events generated by the sandbox entity:

```
sensuctl event list
```

```
    Entity        Check                    Output                    Status  Silenced      Timestamp
_____ _____ _____
_____ _____ _____
    sensu-go-sandbox   keepalive   Keepalive last sent from sensu-go-sandbox at 2019-01-24 21:29:06 +0000 UTC      0   false
    2019-01-24 21:29:06 +0000 UTC
```

The sensu-go-sandbox keepalive event has status 0, which means the agent is in an OK state and can communicate with the Sensu backend.

You can also see the event and the entity in the Sensu dashboard. Log in to the dashboard with these pre-set admin credentials: username `admin` and password `P@ssw0rd!` .

Lesson #2: Pipe keepalive events into Slack

Now that you know the sandbox is working properly, let's get to the fun stuff: creating a workflow. In this lesson, you'll create a workflow that sends keepalive alerts to Slack.

NOTE: If you'd rather not create a Slack account, you can skip ahead to *Lesson #3*.

1. Get your Slack webhook URL

Create a Slack workspace (or use an existing workspace, if you're already a Slack admin).

Then, visit `YOUR-WORKSPACE-NAME.slack.com/services/new/incoming-webhook` . Follow the steps to add the *Incoming WebHooks* integration and save your webhook. Your webhook channel and URL will be listed under Integration Settings — you'll need both later in this lesson.

2. Register the Sensu Slack handler asset

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins. In this lesson, we'll use the Sensu Slack handler asset to power a `slack` handler.

Use sensuctl to register the Sensu Slack handler asset.

```
sensuctl asset create sensu-slack-handler --url
"https://assets.bonsai.sensu.io/3149de09525d5e042a83edbb6eb46152b02b5a65/sensu-
slack-handler_1.0.3_linux_amd64.tar.gz" --sha512
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8"
```

You should see a confirmation message from sensuctl.

```
Created
```

The `sensu-slack-handler` asset is now ready to use with Sensu. Use sensuctl to see the complete asset definition.

```
sensuctl asset info sensu-slack-handler --format yaml
```

PRO TIP: *You can use resource definitions to create and update resources (like assets) using* `sensuctl create --file filename.yaml` . *See the sensuctl docs for more information.*

3. Create a Sensu Slack handler

Open the `sensu-slack-handler.json` handler definition provided with the sandbox in your preferred text editor. Edit the definition to include your Slack channel, webhook URL, and the `sensu-slack-handler` asset.

NOTE: *If you aren't sure how to open the handler and edit the definition, try these Vi/Vim gist instructions.*

```
"env_vars": [
```

```
    "KEEPALIVE_SLACK_WEBHOOK=https://hooks.slack.com/services/AAA/BBB/CCC",
    "KEEPALIVE_SLACK_CHANNEL=#monitoring"
  ],
  "runtime_assets": ["sensu-slack-handler"]
```

Now you can create a Slack handler named `keepalive` to process keepalive events.

```
sensuctl create --file sensu-slack-handler.json
```

Use sensuctl to see available event handlers — in this case, you'll only see the `keepalive` handler you just created.

```
sensuctl handler list
```

Name	Type	Timeout	Filters	Mutator	Execute	Environment Variables	Assets
keepalive	pipe	0			RUN: /usr/local/bin/sensu-slack-handler -c "${KEEPALIVE_SLACK_CHANNEL}" -w "${KEEPALIVE_SLACK_WEBHOOK}"	KEEPALIVE_SLACK_WEBHOOK=https://hooks.slack.com/services/AAA/BBB/CCC,KEEPALIVE_SLACK_CHANNEL=#monitoring	sensu-slack-handler

Sensu monitoring events should begin arriving in your Slack channel, indicating that the sandbox entity is in an OK state.

4. Filter keepalive events

Now that you're generating Slack alerts, let's reduce the potential for alert fatigue by adding a filter that sends only warning, critical, and resolution alerts to Slack.

To accomplish this, you'll interactively add the built-in is_incident event filter to the keepalive handler, which will make sure you only receive alerts when the sandbox entity fails to send a keepalive event.

```
sensuctl handler update keepalive
```

The first prompt will be for environment variables. Just press `return` to continue. The second prompt is for the filters selection — enter `is_incident` to apply the is_incident event filter.

```
? Filters: is_incident
```

For each of the mutator, timeout, type, runtime assets, and command prompts, just press `return`.

Use sensuctl to confirm that the keepalive handler now includes the is_incident event filter:

```
sensuctl handler info keepalive
```

```
=== keepalive
Name:          keepalive
Type:          pipe
Timeout:          0
Filters:          is_incident
Mutator:
Execute:          RUN:   sensu-slack-handler -c "${KEEPALIVE_SLACK_CHANNEL}" -w
"${KEEPALIVE_SLACK_WEBHOOK}"
Environment Variables:  KEEPALIVE_SLACK_WEBHOOK=https://hooks.slack.com/services/AAA/BBB/CCC,
KEEPALIVE_SLACK_CHANNEL=#monitoring
Runtime Assets:     sensu-slack-handler
```

With the event filter in place, you should no longer receive messages in your Slack channel every time the sandbox entity sends a keepalive event.

Let's stop the agent and confirm that you receive the expected warning message.

```
sudo systemctl stop sensu-agent
```

After a couple minutes, you should see a warning message in your Slack channel informing you that the sandbox entity is no longer sending keepalive events.

Start the agent to resolve the warning.

```
sudo systemctl start sensu-agent
```

Lesson #3: Automate event production with the Sensu agent

So far, you've used the Sensu agent's built-in keepalive feature, but in this lesson, you'll create a check that automatically produces workload-related events. Instead of sending alerts to Slack, you'll store event data with InfluxDB and visualize it with Grafana.

1. Make sure the Sensu agent is running

```
sudo systemctl restart sensu-agent
```

2. Install Nginx and the Sensu HTTP Plugin

You'll use the Sensu HTTP Plugin to monitor an Nginx server running on the sandbox.

First, install the EPEL release package:

```
sudo yum install -y epel-release
```

Then, install and start Nginx:

```
sudo yum install -y nginx && sudo systemctl start nginx
```

Make sure it's working:

```
curl -I http://localhost:80
```

```
HTTP/1.1 200 OK
...
```

Then install the Sensu HTTP Plugin:

```
sudo sensu-install -p sensu-plugins-http
```

You'll use the `metrics-curl.rb` plugin. Test its output with:

```
/opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -u "http://localhost"
```

```
...
sensu-go-sandbox.curl_timings.http_code 200 1535670975
```

3. Create an InfluxDB pipeline

Now, let's create the InfluxDB pipeline to store these metrics and visualize them with Grafana. To create a pipeline to send metric events to InfluxDB, start by registering the Sensu InfluxDB handler asset.

```
sensuctl asset create sensu-influxdb-handler --url
"https://assets.bonsai.sensu.io/b28f8719a48aa8ea80c603f97e402975a98cea47/sensu-
influxdb-handler_3.1.2_linux_amd64.tar.gz" --sha512
"612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9b
1257bb479676bc155b24e21bf93c722b812b0f15cb3bd"
```

You should see a confirmation message from sensuctl.

```
Created
```

The `sensu-influxdb-handler` asset is now ready to use with Sensu. Use sensuctl to see the complete

asset definition.

```
sensuctl asset info sensu-influxdb-handler --format yaml
```

Open the `influx-handler.json` handler definition provided with the sandbox, and edit the `runtime_assets` attribute to include the `sensu-influxdb-handler` asset.

```
"runtime_assets": ["sensu-influxdb-handler"]
```

Now you can use sensuctl to create the `influx-db` handler:

```
sensuctl create --file influx-handler.json
```

Use sensuctl to confirm that the handler was created successfully.

```
sensuctl handler list
```

The `influx-db` handler should be listed. If you completed <u>lesson #2</u>, you'll also see the `keepalive` handler.

4. Create a check to monitor Nginx

The `curl_timings-check.json` file provided with the sandbox will create a service check that runs the `metrics-curl.rb` check plugin every 10 seconds on all entities with the `entity:sensu-go-sandbox` subscription and sends events to the InfluxDB pipeline. The `metrics-curl.rb` plugin is already included as the value of the command field in `curl_timings-check.json` — you just need to create the file:

```
sensuctl create --file curl_timings-check.json
```

```
sensuctl check list
```

Name	Command	Interval	Cron	Timeout	TTL	Subscriptions	Handlers	Assets	Hooks	Publish?	Stdin?	Metric Format	Metric Handlers
curl_timings	/opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -u "http://localhost"	10		0	0	entity:sensu-go-sandbox				true	false	graphite_plaintext	influx-db

This check specifies a metrics handler and metric format. In Sensu Go, metrics are a core element of the data model: you can build pipelines to handle metrics separately from alerts. This allows you to customize your monitoring workflows to get better visibility and reduce alert fatigue.

After about 10 seconds, you can see the event produced by the entity:

```
sensuctl event info sensu-go-sandbox curl_timings --format json | jq .
```

```
...
  "history": [
    {
      "status": 0,
      "executed": 1556472457
    },
  ],
  "output": "sensu-go-sandbox.curl_timings.time_total 0.005 1556472657\n...",
  ...
  "output_metric_format": "graphite_plaintext",
  "output_metric_handlers": [
    "influx-db"
  ],
...
```

Because the check definition specified a metric format of `graphite_plaintext`, the Sensu agent will treat the output of the check command as Graphite-formatted metrics and translate them into a set of Sensu-formatted metrics (not shown in the output). These metrics are then sent to the InfluxDB handler, which reads Sensu-formatted metrics and converts them to a format InfluxDB accepts.

NOTE: Metric support isn't limited to Graphite! The Sensu agent can extract metrics in multiple line protocol formats, including Nagios performance data.

5. See the HTTP response code events for Nginx in Grafana.

Log in to Grafana with username: `admin` and password: `admin` .You should see a graph of live HTTP response codes for Nginx.

Now, if you turn Nginx off, you should see the impact in Grafana:

```
sudo systemctl stop nginx
```

Start Nginx:

```
sudo systemctl start nginx
```

6. Automate disk usage monitoring for the sandbox

Now that you have an entity set up, you can add more checks.For example, let's say you want to monitor disk usage on the sandbox.

First, install the plugin:

```
sudo sensu-install -p sensu-plugins-disk-checks
```

Test the plugin:

```
/opt/sensu-plugins-ruby/embedded/bin/metrics-disk-usage.rb
```

```
sensu-core-sandbox.disk_usage.root.used 2235 1534191189
sensu-core-sandbox.disk_usage.root.avail 39714 1534191189
...
```

Then create the check using sensuctl and the `disk_usage-check.json` file included with the sandbox, assigning it to the `entity:sensu-go-sandbox` subscription and the InfluxDB pipeline:

```
sensuctl create --file disk_usage-check.json
```

You don't need to make any changes to disk_usage-check.json before running `sensuctl create --file disk_usage-check.json`.

You should see the check working on the <u>dashboard Entity page</u> and via sensuctl:

```
sensuctl event list
```

Now, you should be able to see disk usage metrics for the sandbox in Grafana: <u>reload your Grafana tab to show the Sensu Go Sandbox Combined</u>.

You made it! You're ready for the next level of Sensu-ing.

Before you move on, take a moment to remove the virtual machine and resources installed during this sandbox lesson. Press `CTRL` + `D` to exit the sandbox. Then run:

```
vagrant destroy
```

Now you can continue exploring Sensu with a clean slate. Here are some resources to help continue your journey:

- ¬ Try another <u>lesson in the Sensu sandbox</u>
- ¬ <u>Install Sensu Go</u>
- ¬ <u>Collect StatsD metrics</u>
- ¬ <u>Create a read-only user</u>

Collect Prometheus metrics with Sensu

The Sensu Prometheus Collector is a check plugin that collects 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.

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.

This guide uses 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 Sensu Prometheus Collector will then 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.

Set up

Install and configure Prometheus

Download and extract Prometheus:

```
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 (your result may vary slightly from this example):

```
ps -ef | grep "[p]rometheus"
vagrant    7647   3937   2 22:23 pts/0       00:00:00 ./prometheus --
config.file=prometheus.yml
```

Install and configure Sensu Go

Follow the RHEL/CentOS install instructions for the Sensu backend, the Sensu agent, and sensuctl.

Add an `app_tier` subscription to `/etc/sensu/agent.yml`:

```
subscriptions:
  - "app_tier"
```

Restart the Sensu agent to apply the configuration change:

```
sudo systemctl restart sensu-agent
```

Ensure Sensu services are running:

```
systemctl status sensu-backend
systemctl status sensu-agent
```

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:

```
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 dashboard:

```
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:

```yaml
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 pipeline

Create a Sensu InfluxDB handler asset

Put the following asset definition in a file called `asset_influxdb` :

YML

```yaml
type: Asset
api_version: core/v2
metadata:
  name: sensu-influxdb-handler
  namespace: default
spec:
  sha512:
612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9b1
257bb479676bc155b24e21bf93c722b812b0f15cb3bd
  url:
https://assets.bonsai.sensu.io/b28f8719a48aa8ea80c603f97e402975a98cea47/sensu-
influxdb-handler_3.1.2_linux_amd64.tar.gz
```

JSON

```json
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-influxdb-handler",
    "namespace": "default"
  },
  "spec": {
    "sha512":
"612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9b
1257bb479676bc155b24e21bf93c722b812b0f15cb3bd",
    "url":
"https://assets.bonsai.sensu.io/b28f8719a48aa8ea80c603f97e402975a98cea47/sensu-
influxdb-handler_3.1.2_linux_amd64.tar.gz"
  }
}
```

Create a Sensu handler

Put the following handler definition in a file called `handler` :

YML

```yaml
type: Handler
api_version: core/v2
metadata:
  name: influxdb
  namespace: default
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
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "influxdb",
    "namespace": "default"
  },
  "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"
    ]
  }
}
```

PRO TIP: `sensuctl create -f` *also accepts files that contain multiple resources' definitions.*

Use `sensuctl` to add the handler and the asset to Sensu:

```
sensuctl create --file handler --file asset_influxdb
```

Collect Prometheus metrics with Sensu

Create a Sensu Prometheus Collector asset

Put the following handler definition in a file called `asset_prometheus`:

YML

```
type: Asset
api_version: core/v2
metadata:
  name: sensu-prometheus-collector
  namespace: default
spec:
  url:
https://assets.bonsai.sensu.io/ef812286f59de36a40e51178024b81c69666e1b7/sensu-
prometheus-collector_1.1.6_linux_amd64.tar.gz
  sha512:
a70056ca02662fbf2999460f6be93f174c7e09c5a8b12efc7cc42ce1ccb5570ee0f328a2dd8223f506df
3b5972f7f521728f7bdd6abf9f6ca2234d690aeb3808
```

JSON

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-prometheus-collector",
    "namespace": "default"
  },
  "spec": {
    "url":
"https://assets.bonsai.sensu.io/ef812286f59de36a40e51178024b81c69666e1b7/sensu-
prometheus-collector_1.1.6_linux_amd64.tar.gz",
    "sha512":
```

```
"a70056ca02662fbf2999460f6be93f174c7e09c5a8b12efc7cc42ce1ccb5570ee0f328a2dd8223f506d
f3b5972f7f521728f7bdd6abf9f6ca2234d690aeb3808"
    }
}
```

Add a Sensu check to complete the pipeline

Create the following check definition in a file called `check` :

YML

```yml
type: CheckConfig
api_version: core/v2
metadata:
  name: prometheus_metrics
  namespace: default
spec:
  command: "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-query
up"
  handlers:
  - influxdb
  interval: 10
  publish: true
  output_metric_format: influxdb_line
  output_metric_handlers: []
  subscriptions:
  - app_tier
  timeout: 0
  runtime_assets:
  - sensu-prometheus-collector
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "prometheus_metrics",
    "namespace": "default"
  },
```

```
  "spec": {
    "command": "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
query up",
    "handlers": [
      "influxdb"
    ],
    "interval": 10,
    "publish": true,
    "output_metric_format": "influxdb_line",
    "output_metric_handlers": [],
    "subscriptions": [
      "app_tier"
    ],
    "timeout": 0,
    "runtime_assets": [
      "sensu-prometheus-collector"
    ]
  }
}
```

Use `sensuctl` to add the check to Sensu:

```
sensuctl create --file check --file asset_prometheus
```

Open the Sensu dashboard to see 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 see the metric event data using sensuctl.

```
sensuctl event list
   Entity        Check                              Output                        Status  Silenced       Timestamp
 ─────────────── ─────────────────────
──────────────────────────────────────────────────────────────────────── ─────────
─────────── ─────────────────────────────────────
sensu-centos   keepalive        Keepalive last sent from sensu-centos at 2019-02-12 01:01:37 +0000 UTC      0   false
2019-02-12 01:01:37 +0000 UTC
```

sensu-centos   prometheus_metrics   up,instance=localhost:9090,job=prometheus value=1 1549933306   0   false
2019-02-12 01:01:46 +0000 UTC

Visualize metrics with Grafana

Configure a dashboard in Grafana

Download the Grafana dashboard configuration file from the Sensu docs:

```
wget 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: login 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.You should see a graph with initial metrics, similar to:
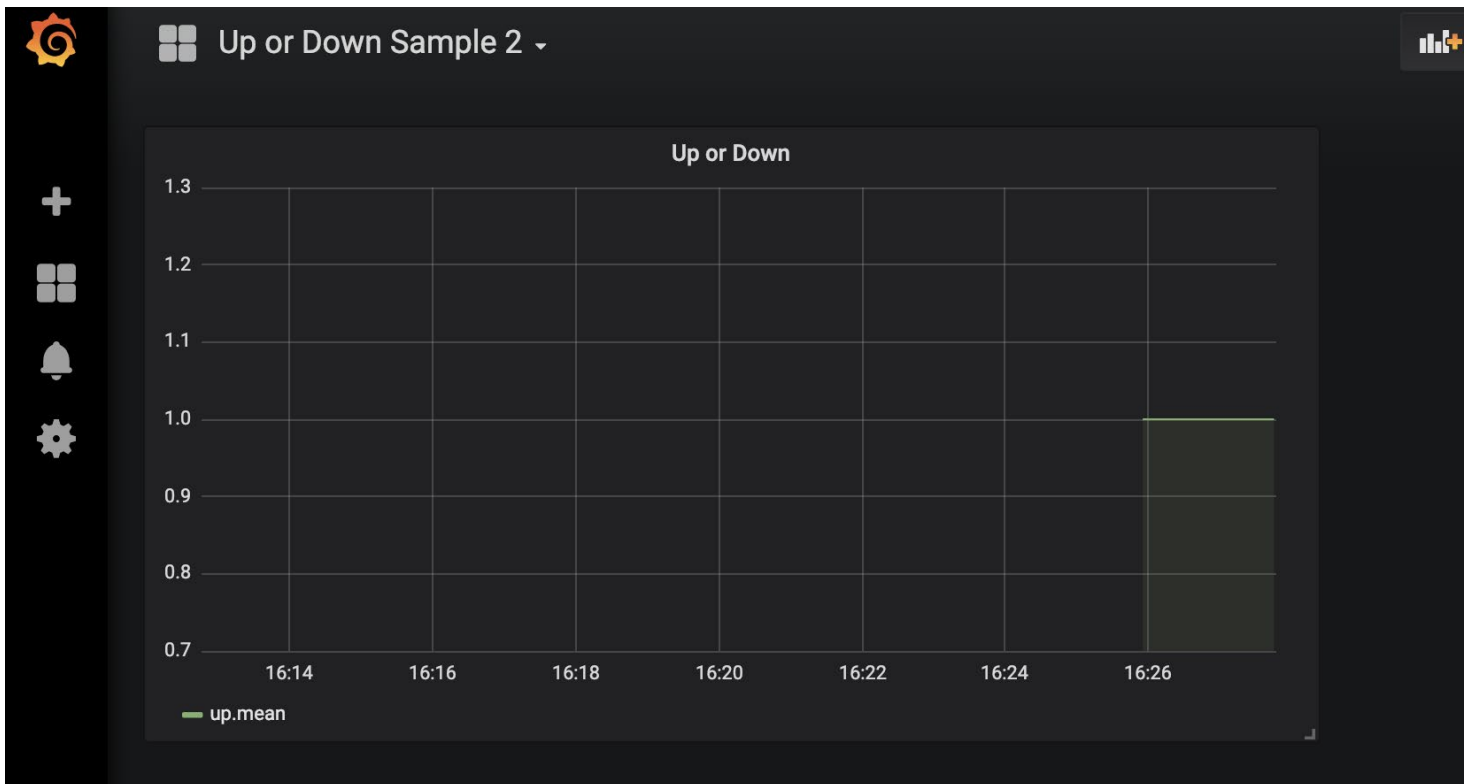
Next steps

You should now have a working set-up with Prometheus scraping metrics. The Sensu Prometheus Collector runs via a Sensu check and collects metrics from the Prometheus API. The metrics are handled by the InfluxDB handler, sent to InfluxDB, and visualized by a Grafana dashboard.

You can plug the Sensu Prometheus Collector into your Sensu ecosystem. 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.

Reference