Sensu Go

Contents

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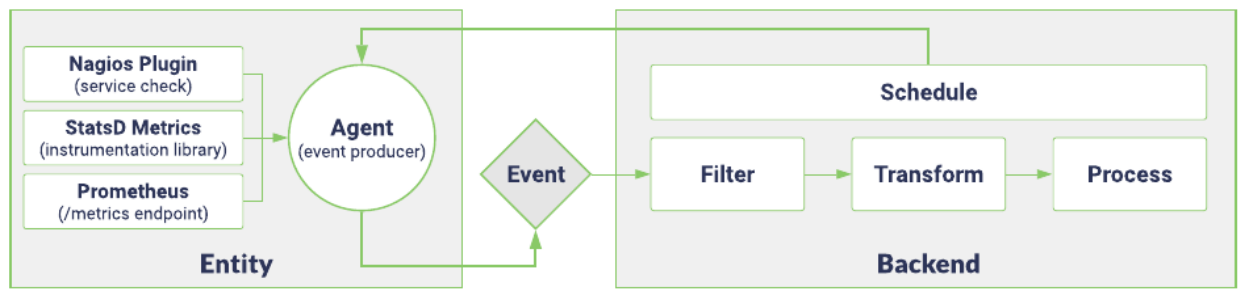
Sensu Observability Pipeline

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

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

Get started now and feel the #monitoringlove.

Filtered, context-rich alerts that improve incident response

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

Plugins that extend functionality and integrate with existing workflows

Sensu's open architecture integrates with the tools and services you already use, like Ansible, EC2, InfluxDB, PagerDuty, Puppet, Rundeck, Saltstack, Slack, and Sumo Logic.

Check out our supported integrations, search for more plugins in Bonsai, the Sensu asset hub, or write your own Sensu plugins in any language.

Automate with agent registration-deregistration and check

subscriptions

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

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

Built-in support for industry-standard tools

Know what's going on everywhere in your system. Sensu supports industry-standard metric formats like Nagios performance data, Graphite plaintext protocol, InfluxDB line protocol, OpenTSDB data specification, Prometheus Exposition Text Format, and StatsD metrics. Use the Sensu agent to collect metrics alongside check results, then use the Sensu observability pipeline to route observation data to a time-series database like InfluxDB.

Intuitive API with command line and web interfaces

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

Commercial software based on open core

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

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

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

Sensu Go release notes

Versioning

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

Upgrading

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

6.1.4 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 6.1.4.

FIXES:

- Fixed a bug that could cause a panic in the backend entity API.
- The agent asset fetching mechanism now respects HTTP proxy environment variables when `trusted-ca-file` is configured.
- When an asset artifact retrieved by the agent does not match the expected checksum, the logged error now includes the size of the retrieved artifact and more clearly identifies the expected and actual checksums.

6.1.3 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 6.1.3.

FIXES:

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

6.1.2 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 6.1.2.

FIXES:

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

6.1.1 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 6.1.1.

FIXES:

- ⌐ (Commercial feature) `sensuctl prune` now properly handles hooks when pruning resources.
- ⌐ (Commercial feature) Fixed a bug that returned incorrect `!=` results for label selectors when no labels were defined.
- ⌐ (Commercial feature) In the web UI, fixed a bug that could cause a GraphQL `no claims` error when a user's access token was no longer valid instead of displaying the sign-out dialog

window.

- (Commercial feature) In the web UI, form validation for subscription names now matches allowed values.

- Fixed a bug that prevented sensu-agent from shutting down correctly.

- Entities are now properly created using `sensuctl create`.

- Per-entity subscriptions now persist with PATCH requests.

- Any user with `update` permissions for a resource can now make PATCH requests for that resource.

- HookConfig can now be exported via `sensuctl dump`. Also, `sensuctl dump` now properly logs API errors.

- eventd errors now include additional context for debugging.

6.1.0 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 6.1.0.

NEW FEATURES:

- (Commercial feature) Added support for custom secrets engine paths in Vault secrets.

- (Commercial feature) In the web UI, added new search functionality, with improved syntax and suggestions.

- (Commercial feature) Added `strict` attribute to the PostgresConfig type to help debug incorrect configurations and database permissions.

- (Commercial feature) Added `batch_buffer`, `batch_size`, and `batch_workers` attributes

to the PostgresConfig type so operators can optimize PostgreSQL latency and throughput.

- ⌐ (Commercial feature) Added TLS configuration to the cluster resource so you can specify additional CA certificates and insecure mode.

- ⌐ (Commercial feature) Added a `types` query parameter for listing authentication providers and secrets providers via the API.

- ⌐ (Commercial feature) Added the Sensu SaltStack Enterprise Handler for launching SaltStack Enterprise Jobs for automated remediation.

- ⌐ (Commercial feature) The Alpine-based Docker image now has multi-architecture support with support for the linux/386, linux/amd64, linux/arm64, linux/arm/v6, linux/arm/v7, linux/ppc64le, and linux/s390x platforms.

- ⌐ The backend flag `--api-request-limit` is now available to configure the maximum API request body size (in bytes).

- ⌐ In the REST API, most configuration resources now support the PATCH method for making updates.

- ⌐ Added new handler and check plugins: Sensu Go Elasticsearch Handler, Sensu Rundeck Handler, and Sensu Kubernetes Events Check.

IMPROVEMENTS:

- ⌐ (Commercial feature) Improved logging for OIDC authentication providers. Also added `disable_offline_access` OIDC spec attribute, which provides a workaround for authorization servers that do not support the `offline_access` scope.

- ⌐ (Commercial feature) Added indexed field and label selectors to the PostgreSQL event store to improve performance for PostgreSQL event store queries with field and label selectors.

- ⌐ Added Prometheus transformer for extracting metrics from check output using the Prometheus Exposition Text Format.

- ⌐ Added the `output_metric_tags` attribute for checks so you can apply custom tags to enrich metric points produced by check output metric extraction.

- ⌐ A warning is now logged when you request a dynamic runtime asset that does not exist.

- ⌐ The trusted CA file is now used for agent, backend, and sensuctl asset retrieval.

- ⌐ Per-entity subscriptions (such as `entity:entityName`) are always available for agent entities, even you remove subscriptions via the entities API.

- ⌐ Updated the Sensu TimescaleDB Handler to write tags as a JSON object instead of an array of objects, which facilitates tags queries.

- ⌐ Updated the Sensu Go Data Source for Grafana plugin to support using API keys, fetching resources from all namespaces, using Sensu's built-in resposne filtering, grouping aggregation

results by attribute, and number of <u>other improvements</u>.

FIXES:

- ˥ (<u>Commercial feature</u>) Fixed a bug in `sensuctl dump` that allowed polymorphic resources (e.g., secrets providers and authentication providers) to dump other providers of the same type.

- ˥ (<u>Commercial feature</u>) Check output is no longer truncated in the event log file when the max output size is set and the PostgreSQL event store is enabled.

- ˥ (<u>Commercial feature</u>) Sensuctl prune now handles multi-file/multi-url input correctly.

- ˥ (<u>Commercial feature</u>) Fixed a bug where PostgreSQL errors could cause the backend to panic.

- ˥ (<u>Commercial feature</u>) Fixed a bug where PostgreSQL would refuse to store event with a negative check status.

- ˥ The backend will no longer start if the web UI TLS configuration is not fully specified.

- ˥ The agent entity is now included in data passed to the STDIN for the command process.

- ˥ Improved check scheduling to prevent stale proxy entity data when using cron or round robin schedulers.

- ˥ Fixed a bug that resulted in incorrect entity listings for agent entities created via the API instead of sensu-agent.

- ˥ When downloading assets, Sensu now closes the response body after reading from it.

- ˥ Fixed a crash in the backend and agent related to JavaScript execution.

6.0.0 release notes

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

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 6.0.0.

BREAKING CHANGES FOR SENSU 6.0:

- The database schema for entities has changed. As a result, after you complete the steps to upgrade to Sensu 6.0 (including running the `sensu-backend upgrade` command), you will not be able to use your database with older versions of Sensu.

- For Sensu Go instances built from source, the web UI is now a standalone product — it is no longer included with the Sensu backend. Visit the Sensu Go Web repository for more information.

- After initial creation, you cannot change your `sensu-agent` entity configuration by modifying the agent's configuration file.

NEW FEATURES:

- (Commercial feature) Sensu now logs a warning when secrets cannot be sent to an agent because mTLS is not enabled.

- (Commercial feature) Added JavaScript functions `sensu.EventStatus`, `sensu.FetchEvent`, and `sensu.ListEvents` to the filter execution environment so you can now query the Sensu event store for other events within the filter namespace.

- (Commercial feature) Docker-only Sensu now binds to the hostname of containers instead of `localhost`. Docker images now set their own default values for environment variables `SENSU_AGENT_API_URL`, `SENSU_BACKEND_API_URL`, `SENSU_BACKEND_ETCD_INITIAL_CLUSTER`, `SENSU_BACKEND_ETCD_ADVERTISE_CLUSTER`, `SENSU_BACKEND_ETCD_INITIAL_ADVERTISE_PEER_URLS`, `SENSU_BACKEND_ETCD_LISTEN_CLIENT_URLS`, and `ETCD_LISTEN_PEER_URLS`.

- (Commercial feature) Added Linux packages for 386; armv5, armv6, and armv7; MIPS hard float, MIPS LE hard float, and MIPS 64 LE hard float; ppc64le; and s390x architectures. Review the supported platforms page for a complete list of Sensu's supported platforms.

- Added Sensu query expression `sensu.CheckDependencies`.

- Added binary-only distributions for FreeBSD `armv5`, `armv6`, and `armv7` and Linux `ppc64le` and `s390x`.

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

IMPROVEMENTS:

- (Commercial feature) Added support for the `memberOf` attribute in Active Directory (AD).

- (Commercial feature) Added more descriptive information for errors in the federated web UI.

- The `dead` and `handleUpdate` methods in keepalived now use `EntityConfig` and `EntityState` respectively.

-

- The `dead()` and `createProxyEntity()` methods in eventd now use `corev3.EntityConfig` and `corev3.EntityState`.

- Agent entity updates now ignore state-related fields.

- You can now manage Sensu agent configuration via the HTTP API.

- For sysvinit services, Sensu now passes users' secondary groups (that is, groups other than the Sensu user group) to `chroot`, which gives the Sensu agent and backend access to the file access writes that are granted to the secondary groups.

- Output of `sensuctl asset add` now includes help for using the runtime asset.

- For role bindings and cluster role bindings, `subjects.name` values can now include unicode characters, and `roleRef.type` and `subjects.type` values are now automatically capitalized.

- Improved logging for the agent WebSocket connection.

- Improved the wording of the secret provider error message.

- Fewer keys in etcd are now stored for agents.

- Keepalive and round robin scheduling leases are now dealt with more efficiently.

- Upgraded Go version from 1.13.7 to 1.13.15.

- Upgraded etcd version from 3.3.17 to 3.3.22.

FIXES:

- (Commercial feature) Label selectors now work as expected with multiple requirements for events.

- (Commercial feature) Fixed an issue that prevented broken secrets providers from surfacing their errors.

- (Commercial feature) Fixed a bug for PostgreSQL datastores that could prevent GraphQL from retrieving all pages when fetching events in a namespace with more than 1000 total events, resulting in an unexpected error.

- (Commercial feature) Fixed a bug that could cause the backend to panic in case of PostgreSQL errors.

- Sensu now logs and returns and error if it cannot find a mutator.

- Errors produced in the agent by assets, check validation, token substitution, and event unmarshaling are logged once again.

- The User-Agent header is now set only upon on new client creation rather than upon each request.

- When the Sensu agent cannot parse the proper CA certificate path, Sensu logs this in the error message.

- Fixed a bug where highly concurrent event filtering could result in a panic.

- Fixed a bug where nil labels or annotations in an event filtering context would require you to explicitly check whether the annotations or labels are undefined.With this fix, labels and annotations are always defined (although they may be empty).

- Fixed the log entry field for the check's name in schedulerd.

5.21.5 release notes

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

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 5.21.5.

FIXES:

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

5.21.4 release notes

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

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 5.21.4.

FIXES:

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

5.21.3 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 5.21.3.

FIXES:

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

5.21.2 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 5.21.2.

FIXES:

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

5.21.1 release notes

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

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 5.21.1.

IMPROVEMENTS:

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

FIXES:

- ⌐ Fixed a potential web UI crash when fetching events in namespace with 1000 or more events.

- ⌐ Fixed a regression that prevented errors produced in the agent by assets, check validation, token substitution, or event unmarshaling from being logged.

5.21.0 release notes

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

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 5.21.0.

NEW FEATURES:

- ⌐ (<u>Commercial feature</u>) Added <u>entity count and limit</u> for each entity class in the tabular title in the response for `sensuctl license info` (in addition to the total entity count and limit).

- ⌐ (<u>Commercial feature</u>) Added Linux amd64 OpenSSL-linked binaries for the Sensu agent and backend, with accompanying `--require-fips` and `--require-openssl` flags for the <u>agent</u> and <u>backend</u>.

- ⌐ Added `sensuctl user hash-password` command to generate password hashes.

- ⌐ Added the ability to reset passwords via the backend API and `sensuctl user reset-password` .

IMPROVEMENTS:

- ⌐ Changed the <u>default log level for</u> `sensu-agent` to `info` (instead of `warn`).

FIXES:

- ⌐ The password verification logic when running `sensuctl user change-password` is now included in the backend API rather than sensuctl.

- ⌐ Errors in publishing proxy check requests no longer block scheduling for other entities.

- ⌐ Using the `--chunk-size` flag when listing namespaces in sensuctl now works properly.

- ⌐ The agent no longer immediately exits in certain scenarios when components are disabled.

- ⌐ Fixed a bug that could cause a GraphQL query to fail when querying a namespace that contained event data in excess of 2 GB.

5.20.2 release notes

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

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 5.20.2.

NEW FEATURES:

- ⌐ The API request log now includes the username.

FIXES:

- ⌐ <u>(Commercial feature)</u> <u>Process discovery in the agent</u> is temporarily disabled.

- ⌐ The system's libc_type attribute is now properly populated for Ubuntu entities.

- ⌐ Single-letter subscriptions are now allowed.

- ⌐ Subjects are now validated in RBAC role binding and cluster role binding.

- ⌐ <u>Sensuctl command</u> assets can now be retrieved and installed from Bonsai.

5.20.1 release notes

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

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

Read the upgrade guide to upgrade Sensu to version 5.20.1.

FIXES:

- ￢ (Commercial feature) Fixes a bug that prevented the federated homepage in the web UI from retrieving the keepalive and event gauges when PostgreSQL was configured as the event datastore.

- ￢ (Commercial feature) The memory_percent and cpu_percent processes attributes are now properly displayed in the web UI.

- ￢ In the web UI, the entity details page no longer displays float type (which applies only for MIPS architectures). Also on entity details pages, the system's libc type is now listed and process names are no longer capitalized.

5.20.0 release notes

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

This release delivers several new features, substantial improvements, and important fixes. One exciting new feature is agent local process discovery to further enrich entities and their events with valuable context. Other additions include a web UI federation view that provides a single pane of glass for all of your Sensu Go clusters and token substitution for assets. And Windows users rejoice! This release includes many Windows agent fixes, as well as agent log rotation capabilities!

Read the upgrade guide to upgrade Sensu to version 5.20.0.

NEW FEATURES:

- ￢ (Commercial feature) Added a `processes` field to the system type to store agent local processes for entities and events and a `discover-processes` flag to the agent configuration flags to populate the `processes` field in entity.system if enabled.

- ¬ (Commercial feature) Added a new resource, `GlobalConfig`, that you can use to customize your web UI configuration.

- ¬ (Commercial feature) Added metricsd to collect metrics for the web UI and the `metrics-refresh-interval` backend configuration flag for setting the interval at which Sensu should refresh metrics.

- ¬ (Commercial feature) Added process and additional system information to the entity details view in the web UI.

- ¬ (Commercial feature) Added a PostgreSQL metrics suite so metricsd can collect metrics about events stored in PostgreSQL.

- ¬ (Commercial feature) Added entity class limits to the license.

- ¬ Added check hook output to event details page in the web UI.

- ¬ Added the sensuctl describe-type command to list all resource types.

- ¬ Added `annotations` and `labels` as backend configuration options.

- ¬ Added token substitution for assets.

- ¬ Added `event.is_silenced` and `event.check.is_silenced` field selectors.

- ¬ Added `Edition` and `GoVersion` fields to version information. In commercial distributions, the `Edition` version attribute is set to `enterprise`

- ¬ Added ability to configure the Resty HTTP timeout. Also, sensuctl now suppresses messages from the Resty library.

IMPROVEMENTS:

- ¬ (Commercial feature) The web UI homepage is now a federated view.

- ¬ You can now increment the log level by sending SIGUSR1 to the sensu-backend or sensu-agent process.

- ¬ License metadata now includes the current entity count and license entity limit.

- ¬ In the web UI, users will receive a notification when they try to delete an event without appropriate authorization.

- ¬ The Windows agent now has log rotation capabilities.

- ¬ Notepad is now the default editor on Windows rather than vi.

FIXES:

- ¬ (Commercial feature) Database connections no longer leak after queries to the cluster health API.

- In the web UI, any leading and trailing whitespace is now trimmed from the username when authenticating.

- The web UI preferences dialog now displays only the first five groups a user belongs to, which makes the sign-out button more accessible.

- In the web UI, the deregistration handler no longer appears as `undefined` on the entity details page.

- You can now escape quotes to express quoted strings in token substitution templates.

- The Windows agent now accepts and remembers arguments passed to `service run` and `service install`.

- The Windows agent now synchronizes writes to its log file, so the file size will update with every log line written.

- The Windows agent now logs to both console and log file when you use `service run`.

5.19.3 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.19.3.

FIXES:

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

- The etcd event store now returns exact matches when retrieving events by entity (rather than prefixed matches).

- `sensu-backend init` now logs any TLS failures encountered during initialization.

- `sensuctl logout` now resets the TLS configuration.

- `env_vars` values can now include the equal sign.

- Error logs now include underlying errors encountered when fetching an asset.

- The log level is now WARNING when an asset is not installed because none of the filters match.

- ⌐ Fixes a bug in the web UI that could cause labels with links to result in a crash.

- ⌐ Fixes a bug in the web UI that could cause the web UI to crash when using an unregistered theme.

- ⌐ Fixes a bug that could cause the backend to crash.

- ⌐ Fixes a bug in multi-line metric extraction that appeared in Windows agents.

- ⌐ Fixes an authentication bug that restarted the sensu-backend when agents disconnected.

- ⌐ Fixes a bug that meant check `state` and `last_ok` were not computed until the second instance of the event.

- ⌐ Fixes a bug that caused messages like "unary invoker failed" to appear in the logs.

- ⌐ Fixes several goroutine leaks.

- ⌐ Fixes a bug that caused the backend to crash when the etcd client received the error "etcdserver: too many requests."

IMPROVEMENTS:

- ⌐ In the web UI, color-blindness modes are now available.

- ⌐ In the web UI, labels and annotations with links to images will now be displayed inline.

- ⌐ Adds a global rate limit for fetching assets to prevent abusive asset retries, which you can configure with the `--assets-rate-limit` and `--assets-burst-limit` flags for the agent and backend.

- ⌐ Adds support for restarting the backend via SIGHUP.

- ⌐ Adds a timeout flag to `sensu-backend init`.

- ⌐ Deprecated flags for `sensuctl silenced update` subcommand have been removed.

5.19.2 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.19.2.

FIXES:

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

IMPROVEMENTS:

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

5.19.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.19.1.

FIXES:

- ⌐ (Commercial feature) Fixed a bug that caused the PostgreSQL store to be enabled too late upon startup, which caused keepalive bugs and possibly other undiscovered bugs.

- ⌐ Keepalives now fire correctly when using the PostgreSQL event store.

- ⌐ Keepalives can now be published via the HTTP API.

- ⌐ `sensu-agent` no longer allows configuring keepalive timeouts that are shorter than the keepalive interval.

- ⌐ Eventd no longer mistakes keepalive events for checks with TTL.

- ⌐ Keepalives now generate a new event universally unique identifier (UUID) for each keepalive failure event.

- ⌐ Agents now correctly reset keepalive switches on reconnect, which fixes a bug that allowed older keepalive timeout settings to persist.

- ⌐ Token substitution templates can now express escape-quoted strings.

- ⌐ The REST API now uses a default timeout of 3 seconds when querying etcd health.

- ⌐ Pipe handlers now must include a command.

- ⌐ The response for `sensu-backend start --help` and `sensu-agent start --help` now

includes the configuration file default path.

- ¬ The system's `libc_type` attribute is now populated on Alpine containers.

5.19.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.19.0.

NEW FEATURES:

- ¬ (Commercial feature) In the web UI, you can now save, recall, and delete filtered searches.

- ¬ (Commercial feature) Added the `matches` substring matching operator for API response, sensuctl, and web UI filtering selectors.

- ¬ (Commercial feature) Added agent and sensuctl builds for Linux architectures: `mips`, `mipsle`, `mips64`, and `mips64le` (hard float and soft float).

- ¬ (Commercial feature) Sensu now automatically applies the `sensu.io/managed_by` label to resources created via `sensuctl create` for use in the `sensuctl prune` alpha feature.

IMPROVEMENTS:

- ¬ (Commercial feature) The health endpoint now includes PostgreSQL health information.

- ¬ Resource metadata now includes the `created_by` field, which Sensu automatically populates with the name of the user who created or last updated each resource.

- ¬ The agent now discovers entity libc type, VM system, VM role, and cloud provider.

- ¬ System type now includes the `float_type` field, which stores the float type the system is using (hard float or soft float).

- ¬ The Bonsai client now logs at the `debug` level rather than the `info` level.

- ¬

- The store can now create wrapped resources.

- <u>Tessen</u> now collects the type of store used for events (`etcd` or `postgres`) and logs numbers of authentication providers, secrets, and secrets providers. Tessen data helps us understand how we can improve Sensu, and all Tessen transmissions are logged locally for complete transparency.

FIXES:

- Fixed a bug where `event.Check.State` was not set for events passing through the pipeline or written to the event log.

- Fixed a bug that allowed the agent to connect to a backend using a nonexistent namespace.

- Fixed a bug that allowed `subscriptions` to be empty strings.

- Corrected the HTTP status codes for unauthenticated and permission denied errors in the REST API.

- Fixed a bug where check history was incorrectly formed when using the PostgreSQL event store.

5.18.1 release notes

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

Read the <u>upgrade guide</u> to upgrade Sensu to version 5.18.1.

FIXES:

- (<u>Commercial feature</u>) Fixed a bug that caused SQL migrations to fail on PostgreSQL 12.

- (<u>Commercial feature</u>) Fixed a bug where OIDC login could result in a nil pointer panic.

- Changed to using the gRPC client (rather than the embedded etcd client) to improve reliability and avoid nil pointer panics triggered by shutting down the embedded etcd client.

- The Sensu backend no longer hangs indefinitely if a file lock for the asset manager cannot be obtained. Instead, the backend returns an error after 60 seconds.

- Fixed a bug that caused sensu-backend to restart when agents disconnected.

- Fixed a bug where the backend would panic on some 32-bit systems.

5.18.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.18.0.

IMPROVEMENTS:

- ⌐ The `event.entity.entity_class` value now defaults to `proxy` for `POST /events` requests.

- ⌐ If you use the events API to create a new event with an entity that does not already exist, the sensu-backend will automatically create a proxy entity when the event is published.

- ⌐ Sensuctl now accepts Bonsai asset versions that include a prefix with the letter `v` (for example, `v1.2.0`).

- ⌐ The version API now retrieves the Sensu agent version for the Sensu instance.

- ⌐ Log messages now indicate which filter dropped an event.

- ⌐ Sensu now reads and writes `initializationKey` to and from EtcdRoot, with legacy support (read-only) as a fallback.

- ⌐ Sensu will now check for an HTTP response other than `200 OK` response when fetching assets.

- ⌐ Updated Go version from 1.13.5 to 1.13.7.

FIXES:

- ⌐ (Commercial feature) Label selectors and field selectors now accept single and double quotes to identify strings.

- ⌐ Fixed a bug that prevented wrapped resources from having their namespaces set by the default sensuctl configuration.

- ⌐ Fixed a bug that prevented API response filtering from working properly for the silenced API.

- ⌐

- Improved event payload validation for the events API so that events that do not match the URL parameters on the `/events/:entity/:check` endpoint are rejected.
- Sensuctl now supports the `http_proxy`, `https_proxy`, and `no_proxy` environment variables.
- The `auth/test` endpoint now returns the correct error messages.

5.17.2 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.17.2.

FIXES:

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

5.17.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.17.1.

IMPROVEMENTS:

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

FIXES:

- (Commercial feature) When a silencing entry expires or is removed, it is also removed from the silences view in the web UI.

- Fixed a bug that prevented `.tar` assets from extracting if they contain hardlinked files.

- In the web UI, sorting an event list view no longer clears the selected filters.

5.17.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.17.0.

NEW FEATURES:

- (Commercial feature) Added HTTP API for secrets management, with a built-in `Env` secrets provider and support for HashiCorp Vault secrets management. The secrets provider resource is implemented for checks, mutators, and handlers.

- Added the `keepalive-handlers` agent configuration flag to specify the keepalive handlers to use for an entity's events.

IMPROVEMENTS:

- (Commercial feature) Upgraded the size of the events auto-incremented ID in the PostgreSQL store to a 64-bit variant, which allows you to store many more events and avoids exhausting the sequence.

- (Commercial feature) Initialization via `sensu-backend init` is now implemented for Docker.

- (Commercial feature) UPN binding support has been re-introduced via the `default_upn_domain` configuration attribute.

- In the web UI, labels that contain URLs are now clickable links.

- Added `event.entity.name` as a supported field for the `fieldSelector` query parameter.

- In the web UI, users with implicit permissions to a namespace can now display resources within

that namespace.

- ˥ Explicit access to namespaces can only be granted via cluster-wide RBAC resources.

- ˥ You can now omit the namespace from an event in `HTTP POST /events` requests.

- ˥ Added support for the `--format` flag in the sensuctl command list subcommand.

FIXES:

- ˥ (Commercial feature) Fixed a bug where the event check state was not present when using the PostgreSQL event store.

- ˥ (Commercial feature) Agent TLS authentication does not require a license.

- ˥ Fixed a memory leak in the entity cache.

- ˥ Fixed a bug that prevented `sensuctl entity delete` from returning an error when attempting to delete a non-existent entity.

- ˥ In the web UI, fixed a bug that duplicated event history in the event timeline chart.

- ˥ `sensuctl command` assets installed via Bonsai now use the `sensuctl` namespace.

- ˥ Fixed a bug where failing check TTL events could occur if keepalive failures had already occurred.

5.16.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.16.1.

IMPROVEMENTS

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

FIXES:

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

5.16.0 release notes

December 16, 2019 — The latest release of Sensu Go, version 5.16.0, is now available for download. This is another important release, with many new features, improvements, and fixes. We introduced an initialization subcommand for **new** installations that allows you to specify an admin username and password instead of using a pre-defined default. We also added new backend 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.

Read 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.

Read the upgrade guide to upgrade Sensu to version 5.15.0.

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

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

NEW FEATURES:

- ⌐ (Commercial feature) Added support for federation replicators and the federation cluster registration API and the ability to view resources across clusters in the federation in the web UI.

- ⌐ (Commercial feature) Added MSI and NuGet builds for sensuctl. Also, MSI and NuGet installations now add the bin directory to the system PATH on Windows.

- ⌐ (Commercial feature) Added HTTP DELETE access for the license management API.

- ⌐ Added the APIKey resource, with HTTP API support for POST, GET, and DELETE and sensuctl commands to manage the APIKey resource.

- ⌐ Added support for using API keys for API authentication.

- ⌐ Added support for sensuctl commands to install, execute, list, and delete commands from Bonsai or a URL.

- ⌐ Added support for sensu-backend service environment variables.

- ⌐ Added support for timezones in check `cron` strings.

SECURITY:

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

IMPROVEMENTS:

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

FIXES:

- ⌐ Added better error logging for mutator execution.

- ⌐

- Fixed the order of flap detection weighting for checks.

- Fixed the pprof server so it only binds to localhost.

- Moved `corev2.BonsaiAsset` to `bonsai.Asset` and moved `corev2.OutdatedBonsaiAsset` to `bonsai.OutdatedAsset`.

5.14.2 release notes

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

Read the upgrade guide 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.

Read the upgrade guide to upgrade Sensu to version 5.14.1.

NEW FEATURES:

- ¬ Added Prometheus gauges for check schedulers.

FIXES:

- ¬ (Commercial feature) Sensuctl will not incorrectly warn of entity limits for unlimited licenses.

- ¬ (Commercial feature) `oidc` authentication providers will now properly load on start-up.

- ¬ When opening a Bolt database that is already open, `sensu-agent` will not hang indefinitely.

- ¬ Running `sensuctl dump` for multiple resource types with the output format as YAML will not result in separators being printed to `STDOUT` instead of the specified file.

- ¬ Fixed a crash in sensu-backend (panic: send on closed channel).

5.14.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.14.0.

NEW FEATURES:

- ¬ The web UI now includes an error dialog option that allows users to wipe the application's persisted state (rather than having to manually wipe their local/session storage). This can help in the rare case that something in the persisted state is leading to an uncaught exception.

- ¬ The web UI now respects the system preference for operating systems with support for selecting a preferred light or dark theme.

- ¬ `sensuctl dump` can now list the types of supported resources with `sensuctl dump --types`.

- ¬ The entity resource now includes the `sensu_agent_version` field, which reflects the Sensu Semantic Versioning (SemVer) version of the agent entity.

- ¬ There are two new advanced configuration options for `sensu-backend` using embedded etcd: `etcd-heartbeat-interval` and `etcd-election-timeout`.

IMPROVEMENTS:

- ⌐ (Commercial feature) Added support for mutual TLS authentication between agents and backends.

- ⌐ (Commercial feature) Added support for CRL URLs for mTLS authentication.

- ⌐ (Commercial feature) Support agent TLS authentication is usable with the sensu-backend.

- ⌐ In the web UI, feedback is directed to Discourse rather than the GitHub repository's Issues page to facilitate discussion about feature requests.

- ⌐ In the web UI, when a user lands on a page inside a namespace that no longer exists or they do not have access to, the drawer opens to that namespace switcher to help clarify next steps.

- ⌐ Updated Go version from 1.12.3 to 1.13.1.

FIXES:

- ⌐ (Commercial feature) `sensuctl` on Windows can now create Postgres resources.

- ⌐ (Commercial feature) Fixed a bug that resulted in event metrics being ignored when using the Postgres store.

- ⌐ Fixed a bug that caused checks to stop executing after a network error.

- ⌐ Fixed a bug that prevented `sensuctl create` with `stdin` from working.

- ⌐ Splayed proxy checks are executed every interval (instead of every interval + interval * splay_coverage).

- ⌐ Proxy entity labels and annotations are now redacted in the web UI as expected.

- ⌐ Fixed a bug in the ring that prevented round robin schedules from recovering after quorum loss.

- ⌐ Updated web UI so that unauthorized errors emitted while creating silences or resolving events are caught and a notification is presented to communicate what occurred.

- ⌐ Web UI does not report internal errors when a user attempts to queue an ad hoc check for a keepalive.

- ⌐ Fixed a bug in the web UI that may have prevented users with appropriate roles from resolving events, queuing checks, and creating silenced entries.

- ⌐ Asset builds are not separated into several assets unless the the tabular format is used in `sensuctl asset list`.

- ⌐ The 'flag accessed but not defined' error is corrected in `sensuctl asset outdated`.

5.13.2 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.13.2.

FIXES:

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

5.13.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.13.1.

FIXES:

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

5.13.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.13.0.

NEW FEATURES:

- ¬ Sensuctl now integrates with Bonsai, the Sensu asset hub.Run a single sensuctl command to add an asset to your Sensu cluster (for example, `sensuctl asset add sensu/sensu-pagerduty-handler:1.1.0`).Check for outdated assets (new releases available) with the `outdated` subcommand (for example, `sensuctl asset outdated`).

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

- ¬ Sensuctl now supports loading multiple resource configuration files (for example, checks and handlers) from directories!Sensuctl can also load a file using a URL (for example, `sensuctl create -r -f ./checks` and `sensuctl create -f https://my.blog.ca/sensu-go/check.yaml`).

FIXES:

- ¬ Sensuctl interactive check create and update modes now have `none` for the metric output format as the first highlighted option instead of `nagios-perfdata` .

- ¬ Fixed a bug where silences would not expire on event resolution.

5.12.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.12.0.

IMPORTANT:

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

NEW FEATURES:

- ¬ Operators can now authenticate to Sensu via OpenID Direct Connect (OIDC) using sensuctl. Read the authentication documentation for details.

- ⌐ Added sensu-agent and sensuctl binary builds for FreeBSD.

- ⌐ Added sensuctl dump command to output resources to a file or STDOUT, making it easier to back up your Sensu backends.

- ⌐ Agents can now be configured with a list of executables that are allowed to run as check and hook commands.Read the agent reference for more information.

IMPROVEMENTS:

- ⌐ Assets now support defining multiple builds, reducing the number of individual assets needed to cover disparate platforms in your infrastructure.

- ⌐ (Commercial feature) Namespaces listed in both the web UI and sensuctl are now limited to the namespace to which the user has access.

- ⌐ Hooks now support the use of assets.

- ⌐ The event.check.name field has been added as a supported field selector.

- ⌐ Both the API and sensuctl can now be used to delete assets.

- ⌐ The use of ProtoBuf serialization/deserialization over WebSocket can now be negotiated between agent and backend.

- ⌐ Web UI performance has been improved for deployments with many events and entities.

- ⌐ The resource caches can now rebuild themselves in case of failures.

- ⌐ Event and entity resources can now be created via the API without an explicit namespace.The system will refer to the namespace in the request URL.

- ⌐ Event and entity resources can now be created via the API using the POST verb.

SECURITY:

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

FIXES:

- ⌐ Tabular display of filters via sensuctl now displays `&&` or `||` as appropriate for inclusive and exclusive filters, respectively.

- ⌐ Requesting events from the `GET /events/:entity` API endpoint now returns events only for the specified entity.

- ⌐ Running sensuctl config view without configuration no longer causes a crash.

- ⌐ Creating an entity via sensuctl with the `--interactive` flag now prompts for the entity name

when it is not provided on the command line.

- ⌐ Check hooks with `stdin: true` now receive actual event data on STDIN instead of an empty event.

- ⌐ Some issues with check scheduling and updating have been fixed by refactoring the backend's watcher implementation.

KNOWN ISSUES:

- ⌐ Authentication via OIDC is not yet supported in the web UI.

- ⌐ Deleting an asset will not remove references to said asset. It is the operator's responsibility to remove the asset from the runtime_assets field of the check, hook, filter, mutator, or handler.

- ⌐ Deleting an asset will not remove the tarball or downloaded files from disk. It is the operator's responsibility to clear the asset cache if necessary.

5.11.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.11.1.

FIXES:

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

- ⌐ The agent now sends heartbeats to the backend to detect network failures and reconnect more quickly.

- ⌐ The default handshake timeout for the WebSocket connection negotiation was lowered from 45 to 15 seconds and is now configurable.

5.11.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.11.0.

NEW FEATURES:

- The Sensu web UI now includes a filters page that displays available event filters and filter configuration.

- (Commercial feature) Manage your Sensu event filters from your browser: Sensu's web UI now supports creating, editing, and deleting filters.

- The Sensu web UI now includes a mutators page that displays available mutators and mutator configuration.

- (Commercial feature) Manage your Sensu mutators from your browser: Sensu's web UI now supports creating, editing, and deleting mutators.

- Sensuctl now includes the `sensuctl delete` command, letting you use resource definitions to delete resources from Sensu in the same way as `sensuctl create`.Read the sensuctl reference for more information.

- Assets now include a `headers` attribute to include HTTP headers in requests to retrieve assets, allowing you to access secured assets.Read the asset reference for examples.

- Sensu agents now support the `disable-assets` configuration flag, allowing you to disable asset retrieval for individual agents.Read the agent reference for examples.

- Sensu binary-only distributions are now available as zip files.

IMPROVEMENTS:

- (Commercial feature) The Active Directory authentication provider now supports the `default_upn_domain` attribute, letting you appended a domain to a username when a domain is not specified during login.

- (Commercial feature) The Active Directory authentication provider now supports the `include_nested_groups` attribute, letting you search nested groups instead of just the top-level groups of which a user is a member.

- The `sensuctl config view` command now returns the currently configured username.Read the sensuctl reference for examples.

- The Sensu API now returns the `201 Created` response code for POST and PUT requests instead of `204 No Content`.

- The Sensu backend now provides advanced configuration options for buffer size and worker count of keepalives, events, and pipelines.

- Sensu Go now supports Debian 10.For a complete list of supported platforms, visit the

[platforms page](#).

FIXES:

- The web UI now returns an error when attempting to create a duplicate check or handler.

- Silenced entries are now retrieved from the cache when determining whether an event is silenced.

- The Sensu API now returns an error when trying to delete an entity that does not exist.

- The agent WebSocket connection now performs basic authorization.

- The 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.

Read the [upgrade guide](#) to upgrade Sensu to version 5.10.2.

FIXES:

- Sensu now handles expired licenses as expected.

5.10.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.10.1.

FIXES:

- ¬ The proxy_requests entity_attributes are now all considered when matching entities.

- ¬ Events are now removed when their corresponding entity is deleted.

5.10.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.10.0.

NEW FEATURES:

- ¬ (Commercial feature) The Sensu web UI now includes fast, predictive filtering for viewing checks, entities, events, handlers, and silences, including the ability to filter based on custom labels. Select the filter bar and start building custom views using suggested attributes and values. For more information, read the web UI docs.

- ¬ Free Sensu instances can now delete entities in the web UI entities page. Read the web UI docs to get started using the Sensu web UI.

- ¬ (Commercial feature) Sensu now supports using an external PostgreSQL instance for event storage in place of etcd. PostgreSQL can handle significantly higher volumes of Sensu events, letting you scale Sensu beyond etcd's storage limits. Read the datastore reference for more information.

- ¬ Sensu now includes a cluster ID API endpoint and `sensuctl cluster id` command to return the unique Sensu cluster ID. Read the 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. Read the sensuctl reference for more information and usage examples.

- ¬ Sensu cluster roles can now include permissions to manage your Sensu license using the `license` resource type. Read the RBAC reference to create a cluster role.

¬ The web UI now displays up to 100,000 events and entities on the homepage.

FIXES:

¬ Sensu now optimizes scheduling for proxy checks, solving an issue with high CPU usage when evaluating proxy entity attributes.

¬ The Sensu API now validates resource namespaces and types in request bodies to ensure RBAC permissions are enforced.

¬ Check `state` and `total_state_change` attributes now update as expected based on check history.

¬ Incident and entity links in the web UI homepage now navigate to the correct views.

¬ The web UI now displays non-standard cron statements correctly (for example, `@weekly`).

¬ On sign-in, the web UI now ensures that users are directed to a valid namespace.

¬ In the web UI, code block scrollbars now display only when necessary.

¬ The web UI now displays the handler `timeout` attribute correctly.

¬ When editing resources, the web UI now fetches the latest resource prior to editing.

¬ The web UI now handles array values correctly when creating and editing resources.

5.9.0 release notes

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

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

NEW FEATURES:

¬ The Sensu web UI now includes a handlers page that displays available event handlers and handler configuration. Read the web UI docs to get started using the Sensu web UI.

¬ (Commercial feature) Manage your Sensu event handlers from your browser: Sensu's web UI now supports creating, editing, and deleting handlers. Read the web UI docs to get started using the Sensu web UI.

- (Commercial feature) Sensu now supports event logging to a file using the `event-log-file` and `event-log-buffer-size` configuration flags. You can use this event log file as an input source for your favorite data lake solution. Read the backend reference for more information.

IMPROVEMENTS:

- The Sensu web UI now includes simpler, more efficient filtering in place of filtering using Sensu query expressions.

- Sensu packages are now available for Ubuntu 19.04 (Disco Dingo). Review the supported platforms page for a complete list of Sensu's supported platforms and the installation guide to install Sensu packages for Ubuntu.

FIXES:

- The `occurrences` and `occurrences_watermark` event attributes now increment as expected, giving you useful information about recent events. Read the events reference for an in-depth discussion of these attributes.

- The `/silenced/subscriptions/:subscription` and `/silenced/checks/:check` API endpoints now return silences by check or subscription.

- Sensu now handles errors when seeding initial data, avoiding a panic state.

5.8.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.8.0.

IMPORTANT:

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

IMPROVEMENTS:

- The sensuctl command line tool now supports the `--chunk-size` flag to help you handle large datasets. Read the sensuctl reference for more information.

- Sensu backends now support the `etcd-cipher-suites` configuration option, letting you

specify the cipher suites that can be used with etcd TLS configuration.Read the backend reference for more information.

- ¬ The Sensu API now includes the version API, returning version information for your Sensu instance.Review the API docs for more information.

- ¬ Tessen now collects the numbers of events processed and resources created, giving us better insight into how we can improve Sensu.As always, all Tessen transmissions are logged for complete transparency.Read the Tessen reference for more information.

- ¬ Sensu licenses now include the entity limit attached to your Sensu licensing package.Read the license management docs to learn more about entity limits.

- ¬ Sensu backends now perform better at scale using increased worker pool sizes for events and keepalives.

- ¬ The maximum size of the etcd database and etcd requests is now configurable using the `etcd-quota-backend-bytes` and `etcd-max-request-bytes` backend configuration options. These are advanced configuration options requiring familiarly with etcd.Use with caution.Read the backend reference for more information.

- ¬ Most Sensu resources now use ProtoBuf serialization in etcd.

FIXES:

- ¬ Events produced by checks now execute the correct number of write operations to etcd.

- ¬ API pagination tokens for the 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.

Read the upgrade guide to upgrade Sensu to version 5.7.0.

IMPROVEMENTS:

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

FIXES:

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

5.6.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.6.0.

NEW FEATURES:

- (Commercial feature) Manage your Sensu checks from your browser: Sensu's web user interface now supports creating, editing, and deleting checks.Read the web UI docs to get started using the Sensu web UI.
- (Commercial feature) The Sensu web UI now includes an option to delete entities.
- (Commercial feature) Sensu now supports resource filtering in the Sensu API and sensuctl command line tool.Filter events using custom labels and resource attributes, such as event status and check subscriptions.Review the API docs and sensuctl reference for usage examples.

IMPROVEMENTS:

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

- ⌐ The `sensuctl cluster health` and `sensuctl cluster member-list` commands now include the cluster ID in tabular format.

FIXES:

- ⌐ You can now configure labels and annotations for Sensu agents using command line flags. For example: `sensu-agent start --label example_key="example value"` . Read the agent reference for more examples.

- ⌐ The Sensu web UI now displays the correct checkbox state when no resources are present.

5.5.1 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.5.1.

IMPROVEMENTS:

- ⌐ Sensu agents now support annotations (non-identifying metadata) that help people or external tools interacting with Sensu. Read the agent reference to add annotations in the agent configuration file.

- ⌐ The agent socket event format now supports the `source` attribute to create a proxy entity.

- ⌐ Sensu 5.5.1 is built with Go version 1.12.3.

FIXES:

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

5.5.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.5.0.

NEW FEATURES:

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

IMPROVEMENTS:

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

FIXES:

⌐ The Sensu web UI now displays silences created by LDAP users.

⌐ The web UI now uses a secondary text color for quick-navigation buttons.

5.4.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.4.0.

NEW FEATURES:

⌐ The Sensu dashboard now includes a homepage designed to highlight the most important monitoring data, giving you instant insight into the state of your infrastructure.read the web UI docs for a preview.

⌐ The Sensu API now supports pagination using the `limit` and `continue` query parameters, letting you limit your API responses to a maximum number of objects and making it easier to handle large datasets.Read the API overview for more information.

⌐ Sensu now surfaces internal metrics using the metrics API.Read the metrics API reference for more information.

IMPROVEMENTS:

- Sensu now lets you specify a separate TLS certificate and key to secure the dashboard. Read the backend reference to configure the `dashboard-cert-file` and `dashboard-key-file` 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. Read 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 (for example, `--format yaml --namespace development`).

- The `sensuctl handler delete` and `sensuctl filter delete` commands now correctly delete resources from the currently configured namespace.

- The agent now terminates consistently on SIGTERM and SIGINT.

- In the event of a loss of connection with the backend, the agent now attempts to reconnect to any backends specified in its configuration.

- The dashboard now handles cases in which the creator of a silence is inaccessible.

- The dashboard event details page now displays "-" in the command field if no command is associated with the event.

5.3.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.3.0.

NEW FEATURES:

- Round robin check scheduling lets you distribute check executions evenly over a group of Sensu agents.To enable round robin scheduling, set the `round_robin` check attribute to `true`.Read the checks reference for more information.

- Sensu now provides commercial support for using Microsoft Active Directory as an external authentication provider.Read the authentication guide to configure Active Directory, and check out the getting started guide for more information about commercial features.

- The dashboard now features offline state detection and displays an alert banner if the dashboard loses connection to the backend.

IMPROVEMENTS:

- The agent socket event format now supports the `handlers` attribute, giving you the ability to send socket events to a Sensu pipeline.Read the agent reference to learn more about creating and handling monitoring events using the agent socket.

- Assets now feature improved download performance using buffered I/O.

- The sensuctl CLI now uses a 15-second timeout period when connecting to the Sensu backend.

- The dashboard now includes expandable configuration details sections on the check and entity pages.You can now use the dashboard to review check details like command, subscriptions, and scheduling as well as entity details like platform, IP address, and hostname.

SECURITY:

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

- Sensu now enforces uniform TLS configuration for all three backend components: `apid`, `agentd`, and `dashboardd`.

- The backend no longer requires the `trusted-ca-file` 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.

Read the upgrade guide to upgrade Sensu to version 5.2.1.

FIXES:

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

5.2.0 release notes

February 7, 2019 — The latest release of Sensu Go, version 5.2.0, is now available for download. This release has a ton of exciting content, including the availability of our first enterprise-only features. For more details on these features, read the blog post about Sensu Go 5.2.0. Release 5.2.0 also has some key improvements and fixes: we added support for self-signed CA certificates for sensuctl, check output truncation, and the ability to manage silencing from the event details page in our web UI, to name a few.

Read the upgrade guide to upgrade Sensu to version 5.2.0.

IMPORTANT:

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

NEW FEATURES:

- Our first enterprise-only features for Sensu Go: LDAP authentication, the Sensu ServiceNow handler, and the Sensu JIRA handler.Read the getting started guide.

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

IMPROVEMENTS:

- Sensu now includes support for Debian 8 and 9.Read the installation guide to install Sensu for Debian.

- Sensu's binary-only distribution for Linux is now available for `arm64`, `armv5`, `armv6`, `armv7`, and `386` in addition to `amd64`.Read the installation guide for download links.

- The Sensu dashboard now provides the ability to silence and unsilence events from the Events page.

- The Sensu dashboard Entity page now displays the platform version and deregistration configuration.

- Sensuctl now supports TLS configuration options, allowing you to use a self-signed certificate without adding it to the operating system's CA store, either by explicitly trusting the signer or by disabling TLS hostname verification.Read the sensuctl reference for more information.

- sensuctl now provides action-specific confirmation messages, like `Created`, `Deleted`, and `Updated`.

FIXES:

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

- The Sensu backend now correctly handles errors for missing keepalive events.

- Token-substituted values are now omitted from event data to protect sensitive information.

- Sensu now correctly processes keepalive and check TTL states after entity deletion.

- Sensuctl can now run `sensuctl version` without being configured.

- When disabling users, sensuctl now provides the correct prompt for the action.

5.1.1 release notes

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

support for the Sensu agent and sensuctl for 32-bit Windows systems.

Read the upgrade guide to upgrade Sensu to version 5.1.1.

NEW FEATURES:

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

IMPROVEMENTS:

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

SECURITY:

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

FIXES:

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

5.1.0 release notes

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

Read the upgrade guide to upgrade Sensu to version 5.1.0.

IMPORTANT:

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

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

NEW FEATURES:

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

Read the upgrade guide to upgrade Sensu to version 5.0.1.

FIXED:

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

5.0.0 release notes

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

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

To get started with Sensu Go:

- Download the sandbox.
- Install Sensu Go.
- Get started monitoring server resources.

Get started with Sensu

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

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

Install the commercial distribution of Sensu Go

Sensu's supported platforms include CentOS/RHEL, Debian, Ubuntu, and Windows.

- **Install Sensu Go** and get started for free
- Learn about Sensu's commercial features — all commercial features are available for free in the packaged Sensu Go distribution up to an entity limit of 100
- Discover Sensu dynamic runtime assets on Bonsai, the Sensu asset hub
- Find the Sensu architecture that best meets your needs
- Migrate from Sensu Core and Sensu Enterprise to Sensu Go

Learn Sensu

Watch this video for an 8-minute Sensu Go overview and demo:

We recommend these resources for learning more about Sensu:

- Download the sandbox and create an observability pipeline in your local environment

- Try a live demo of the Sensu web UI

- Sign up for our step-by-step Learn Sensu email course

- Join the Sensu Community Forum on Discourse

PRO TIP: *In addition to the resources listed on this page, try the self-guided Sensu Go Workshop.*

Explore monitoring at scale with Sensu Go

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

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

- Activate your Sensu commercial license

Build Sensu from source (OSS)

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

- Get the Sensu Go binary distribution for your platform

- Visit Sensu Go on GitHub

- Compare OSS and commercial features

- Build from source

Supported platforms and distributions

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

Supported packages

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

Sensu backend

	RHEL/CentOS 6, 7, 8	Debian 8, 9, 10	Ubuntu 14.04 16.04, 18.04 18.10, 19.04 19.10, 20.04
`amd64`	✅	✅	✅

Sensu agent

	RHEL/ CentOS 6, 7, 8	Debian 8, 9, 10	Ubuntu 14.04 16.04 18.04 18.10 19.04 19.10 20.04	Windows 7 and later	Windows Server 2008 R2 and later
`amd64`	✅	✅	✅	✅	✅
`386`	✅	✅	✅	✅	✅

`armv5` `armv6` `armv7`	✅	✅	✅
`ppc64le`	✅	✅	✅
`s390x`	✅	✅	✅

Sensuctl command line tool

	RHEL/ CentOS 6, 7, 8	Debian 8, 9, 10	Ubuntu 14.04 16.04 18.04 18.10 19.04 19.10 20.04	Windows 7 and later	Windows Server 2008 R2 and later
`amd64`	✅	✅	✅	✅	✅
`386`	✅	✅	✅	✅	✅
`armv5` `armv6` `armv7`	✅	✅	✅		
`ppc64le`	✅	✅	✅		
`s390x`	✅	✅	✅		

Docker images

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

Image Name	Base

sensu/sensu	Alpine Linux
sensu/sensu-rhel	Red Hat Enterprise Linux

Binary-only distributions

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

The Sensu web UI is a standalone product — it is not distributed inside the Sensu backend binary. Visit the Sensu Go Web GitHub repository for more information.

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

Linux

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

For binary distributions, we support the following Linux kernels:

- ¬ 3.1.x and later for `armv5`

- ¬ 4.8 and later for `MIPS 64 LE hard float` and `MIPS 64 LE soft float`

- ¬ 2.6.23 and later for all other architectures

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

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

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

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-go_6.1.4_linux_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
sha256sum sensu-go_6.1.4_linux_amd64.tar.gz
```

The result should match the checksum for your platform:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-go_6.1.4_checksums.txt && cat sensu-go_6.1.4_checksums.txt
```

Federal Information Processing Standard (FIPS) Compliance

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

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

Contact Sensu to request builds with FIPS support.

Windows

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

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

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

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

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

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-go_6.1.4_windows_amd64.zip  -OutFile "$env:userprofile\sensu-go_6.1.4_windows_amd64.zip"
```

Generate a SHA-256 checksum for the downloaded artifact:

```
Get-FileHash "$env:userprofile\sensu-go_6.1.4_windows_amd64.zip" -Algorithm SHA256 | Format-List
```

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

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_checksums.txt -OutFile "$env:userprofile\sensu-go_6.1.4_checksums.txt"

Get-Content "$env:userprofile\sensu-go_6.1.4_checksums.txt" | Select-String -Pattern
windows_amd64
```

macOS

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

We support macOS 10.11 and later for binary distributions.

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

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

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

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_darwin_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
shasum -a 256 sensu-go_6.1.4_darwin_amd64.tar.gz
```

The result should match the checksum for your platform:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_checksums.txt && cat sensu-go_6.1.4_checksums.txt
```

Extract the archive:

```
tar -xvf sensu-go_6.1.4_darwin_amd64.tar.gz
```

Copy the executable into your PATH:

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

FreeBSD

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

We support FreeBSD 11.2 and later for binary distributions.

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

Architecture	Formats
`386`	`.tar.gz` | `.zip`
`amd64`	`.tar.gz` | `.zip`
`armv5`	`.tar.gz` | `.zip`
`armv6`	`.tar.gz` | `.zip`
`armv7`	`.tar.gz` | `.zip`

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

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
```

```
go_6.1.4_freebsd_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact:

```
sha256sum sensu-go_6.1.4_freebsd_amd64.tar.gz
```

The result should match the checksum for your platform:

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-go_6.1.4_checksums.txt && cat sensu-go_6.1.4_checksums.txt
```

Solaris

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

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

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

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

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

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-go_6.1.4_solaris_amd64.tar.gz
```

Generate a SHA-256 checksum for the downloaded artifact.

```
sha256sum sensu-go_6.1.4_solaris_amd64.tar.gz
```

The result should match the checksum for your platform.

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_checksums.txt && cat sensu-go_6.1.4_checksums.txt
```

Legacy systems and other platforms

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

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

Build from source

Sensu Go's core is open source software, freely available under an MIT License.Sensu Go instances built from source do not include commercial features such as the web UI homepage.Review the feature comparison matrix to learn more.To build Sensu Go from source, read the contributing guide on GitHub.

Get started with commercial features

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

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

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

Commercial features in Sensu Go

- **Integrate your Sensu observability pipeline with industry-standard tools** like EC2, Jira, ServiceNow, and Sumo Logic with supported integrations and enterprise-tier dynamic runtime assets.

- **Manage resources from your browser**: Use the Sensu web UI to manage events and entities and create, edit, and delete checks, handlers, mutators, silences, and event filters.

- **Control permissions with Sensu role-based access control (RBAC)**, with the option of using Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) for authentication.

- **Use mutual transport layer security (mTLS) authentication** to provide two-way verification of your Sensu agents and backend connections.

- **Protect your sensitive information** with secrets management. Avoid exposing usernames, passwords, and access keys in your Sensu configuration by integrating with HashiCorp Vault or using Sensu's built-in environment variable secrets provider.

- **Manage your monitoring resources across multiple data centers, cloud regions, or providers** and mirror changes to follower clusters with federation. Federation affords visibility into the health of your infrastructure and services across multiple distinct Sensu instances within a single web UI.

- **Use powerful search capabilities** designed for large installations to search Sensu API responses, sensuctl outputs, and Sensu web UI views using custom labels and a wide range of resource attributes. Build event filter expressions with JavaScript execution functions.

- **Log observation data** to a file you can use as an input to your favorite data lake solution.

- **Achieve enterprise-scale event handling** for your Sensu instance with a PostgreSQL event store. Access the PostgreSQL event datastore with the same Sensu web UI, API, and sensuctl processes as etcd-stored events.

- **Get enterprise-class support**: Rest assured that with Sensu support, help is available if you need it. Our expert in-house team offers best-in-class support to get you up and running smoothly.

Review a complete comparison of OSS and commercial features.

Contact us for a free trial

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

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

Get started with commercial features in Sensu Go

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

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

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

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

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
```

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

Use sensuctl to view your license details at any time:

```
sensuctl license info
```

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

- [Set up and manage authentication providers](#)
- [Use dynamic runtime assets to install plugins](#)
- [Manage your Sensu commercial license](#)
- [Log in to your Sensu account](#)
- [Contact Sensu support](#)

Sensu Observability Pipeline

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

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

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

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

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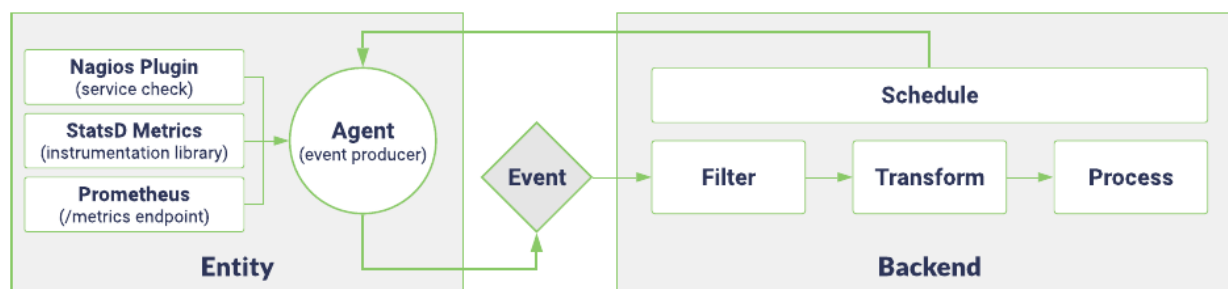
Sensu Observability Pipeline

Entities

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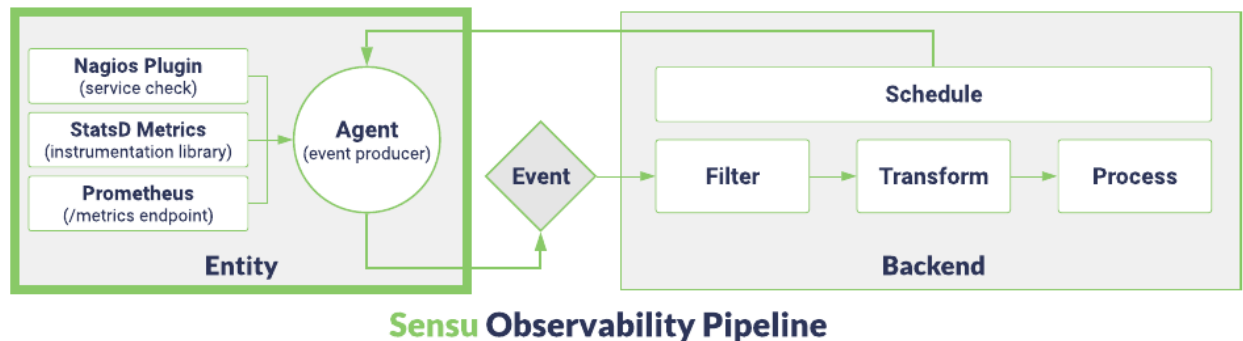
An <u>entity</u> represents anything that needs to be observed or monitored, such as a server, container, or network switch, including the full range of infrastructure, runtime, and application types that compose a complete monitoring environment (from server hardware to serverless functions).Sensu calls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

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

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

Agent entities

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

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

This example shows an agent entity resource definition:

`YML`

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

JSON

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

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

Proxy entities

Proxy entities [formerly known as proxy clients or just-in-time (JIT) clients] are dynamically created entities that Sensu adds to the entity store if an entity does not already exist for a check result. Proxy entities allow Sensu to monitor external resources on systems where you cannot install a Sensu agent, like a network switch or website. Sensu uses the defined check `proxy_entity_name` to create a proxy entity for the external resource.

This example shows a proxy entity resource definition:

YML

```
---
type: Entity
api_version: core/v2
metadata:
  labels:
    proxy_type: website
    sensu.io/managed_by: sensuctl
    url: https://docs.sensu.io
  name: sensu-docs
  namespace: default
spec:
  deregister: false
  deregistration: {}
  entity_class: proxy
```

```
    last_seen: 0
    sensu_agent_version: ""
    subscriptions: null
    system:
      cloud_provider: ""
      libc_type: ""
      network:
        interfaces: null
      processes: null
      vm_role: ""
      vm_system: ""
```

JSON

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

```
        "vm_system": ""
    }
  }
}
```

Proxy entity registration differs from keepalive-based registration because the registration event happens while processing a check result (not a keepalive message).

Read Monitor external resources to learn how to use a proxy entity to monitor a website.

Usage limits

Sensu's usage limits are based on entities.

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

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

- ¬ Entity limit: the maximum number of entities of all classes your license includes. Both agent and proxy entities count toward the overall entity limit.

- ¬ Entity class limits: the maximum number of a specific class of entities (for example, agent or proxy) that your license includes.

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

Use sensuctl or the license API to view your overall entity count and limit.

Entity reference

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

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

Create and manage agent entities

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

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

> **NOTE**: *You cannot modify an agent entity with the* `agent.yml` *configuration file unless you delete the entity.The entity attributes in* `agent.yml` *are used only for initial entity creation unless you delete the entity.*

If you delete an agent entity that you modified with sensuctl, the entities API, or the web UI, it will revert to the original configuration from `agent.yml` .

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

If you change an agent entity's class to `proxy` , the backend will revert the change to `agent` .

Agent entity example

This example shows the resource definition for an agent entity:

YML

```yaml
type: Entity
api_version: core/v2
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
    libc_type: glibc
    vm_system: kvm
    vm_role: host
    cloud_provider: null
    processes:
    - name: Slack
      pid: 1349
      ppid: 0
      status: Ss
      background: true
      running: true
```

```
        created: 1582137786
        memory_percent: 1.09932518
        cpu_percent: 0.3263987595984941
      - name: Slack Helper
        pid: 1360
        ppid: 1349
        status: Ss
        background: true
        running: true
        created: 1582137786
        memory_percent: 0.146866455
        cpu_percent: 0.30897618146109257
        hostname: sensu2-centos
    network:
      interfaces:
      - addresses:
        - 127.0.0.1/8
        - ::1/128
        name: lo
      - addresses:
        - 10.0.2.15/24
        - fe80::26a5:54ec:cf0d:9704/64
        mac: 08:00:27:11:ad:d2
        name: enp0s3
      - addresses:
        - 172.28.128.3/24
        - fe80::a00:27ff:febc:be60/64
        mac: 08:00:27:bc:be:60
        name: enp0s8
    os: linux
    platform: centos
    platform_family: rhel
    platform_version: 7.4.1708
  sensu_agent_version: 1.0.0
  user: agent
```

JSON

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

```json
    "name": "webserver01",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu2-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
              "10.0.2.15/24",
              "fe80::26a5:54ec:cf0d:9704/64"
            ]
          },
          {
            "name": "enp0s8",
            "mac": "08:00:27:bc:be:60",
            "addresses": [
              "172.28.128.3/24",
              "fe80::a00:27ff:febc:be60/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
```

```
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 0.146866455,
        "cpu_percent": 0.308976181461092553
      }
    ]
  },
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "entity:webserver01"
  ],
  "last_seen": 1542667231,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
```

```
            "access_key",
            "secret_key",
            "private_key",
            "secret"
        ]
    }
}
```

Create and manage proxy entities

Proxy entities are dynamically created entities that Sensu adds to the entity store if an entity does not already exist for a check result. Proxy entities allow Sensu to monitor external resources on systems where you cannot install a Sensu agent, like a network switch or website.

You can modify proxy entities via the backend with sensuctl, the entities API, and the web UI.

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

Proxy entity example

This example shows the resource definition for a proxy entity:

YML

```
type: Entity
api_version: core/v2
metadata:
  name: sensu-docs
  namespace: default
  labels: null
spec:
  deregister: false
  deregistration: {}
  entity_class: proxy
  last_seen: 0
  subscriptions:
  - proxy
  system:
```

```
      network:
         interfaces: null
   sensu_agent_version: 1.0.0
```

JSON

```json
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-docs",
    "namespace": "default",
    "labels": null
  },
  "spec": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "proxy",
    "last_seen": 0,
    "subscriptions": [
      "proxy"
    ],
    "system": {
      "network": {
        "interfaces": null
      }
    },
    "sensu_agent_version": "1.0.0"
  }
}
```

Proxy entities and round robin scheduling

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

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

If you use a proxy entity without round robin scheduling, and several agents share the 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 observation data in events that you can use for response and web UI view searches.In contrast to annotations, you can use labels to filter API responses, sensuctl responses, and web UI search views.

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

Agent entity labels

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

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

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

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

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

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, first create a file named `proxy-example.yml` or `proxy-example.json` with an entity definition that includes labels:

YML

```yml
---
type: Entity
api_version: core/v2
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
  sensu_agent_version: 1.0.0
```

JSON

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

```json
    "deregister": false,
    "deregistration": {},
    "entity_class": "proxy",
    "last_seen": 0,
    "subscriptions": [
      "proxy"
    ],
    "system": {
      "network": {
        "interfaces": null
      }
    },
    "sensu_agent_version": "1.0.0"
  }
}
```

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

SHELL

```shell
sensuctl create --file proxy-example.yml
```

SHELL

```shell
sensuctl create --file proxy-example.json
```

To add a label to an existing entity, use sensuctl edit. For example, to add a `proxy_type` label to the `sensu-docs` entity you just created:

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

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

```yaml
type: Entity
api_version: core/v2
metadata:
```

```
    labels:
      url: docs.sensu.io
      proxy_type: website
    name: sensu-docs
    namespace: default
  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. Read Monitor external resources with proxy entities for details about creating a proxy check for a proxy entity.

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 | **YML**<br><br>```
    type: Entity
```<br><br>**JSON** |

```
{
  "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

YML

example

```
api_version: core/v2
```

JSON

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

metadata

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

type	Map of key-value pairs

YML

example	

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

JSON

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

spec

description	Top-level map that includes the entity spec attributes.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

YML

example

```yaml
spec:
  entity_class: agent
  system:
    hostname: sensu2-centos
    os: linux
    platform: centos
    platform_family: rhel
    platform_version: 7.4.1708
    network:
      interfaces:
      - name: lo
        addresses:
        - 127.0.0.1/8
        - "::1/128"
      - name: enp0s3
        mac: '08:00:27:11:ad:d2'
        addresses:
        - 10.0.2.15/24
        - fe80::26a5:54ec:cf0d:9704/64
      - name: enp0s8
        mac: '08:00:27:bc:be:60'
        addresses:
        - 172.28.128.3/24
        - fe80::a00:27ff:febc:be60/64
    arch: amd64
    libc_type: glibc
    vm_system: kvm
    vm_role: host
    cloud_provider: ''
    processes:
    - name: Slack
      pid: 1349
      ppid: 0
      status: Ss
      background: true
      running: true
      created: 1582137786
      memory_percent: 1.09932518
      cpu_percent: 0.3263987595984941
    - name: Slack Helper
      pid: 1360
```

```
        ppid: 1349
        status: Ss
        background: true
        running: true
        created: 1582137786
        memory_percent: 0.146866455
        cpu_percent: 0.30897618146109257
    sensu_agent_version: 1.0.0
    subscriptions:
    - entity:webserver01
    last_seen: 1542667231
    deregister: false
    deregistration: {}
    user: agent
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
```

JSON

```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",
```

```json
          "::1/128"
        ]
      },
      {
        "name": "enp0s3",
        "mac": "08:00:27:11:ad:d2",
        "addresses": [
          "10.0.2.15/24",
          "fe80::26a5:54ec:cf0d:9704/64"
        ]
      },
      {
        "name": "enp0s8",
        "mac": "08:00:27:bc:be:60",
        "addresses": [
          "172.28.128.3/24",
          "fe80::a00:27ff:febc:be60/64"
        ]
      }
    ]
  },
  "arch": "amd64",
  "libc_type": "glibc",
  "vm_system": "kvm",
  "vm_role": "host",
  "cloud_provider": "",
  "processes": [
    {
      "name": "Slack",
      "pid": 1349,
      "ppid": 0,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 1.09932518,
      "cpu_percent": 0.3263987595984941
    },
    {
      "name": "Slack Helper",
      "pid": 1360,
      "ppid": 1349,
```

```
            "status": "Ss",
            "background": true,
            "running": true,
            "created": 1582137786,
            "memory_percent": 0.146866455,
            "cpu_percent": 0.30897618146109257
          }
        ]
      },
      "sensu_agent_version": "1.0.0",
      "subscriptions": [
        "entity:webserver01"
      ],
      "last_seen": 1542667231,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ]
    }
  }
```

Metadata attributes

name	
description	Unique name of the entity, validated with Go regex `\A[\w\.\-]+\z` .
required	true

type	String

YML

```yml
name: example-hostname
```

JSON

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

namespace

description	Sensu RBAC namespace that this entity belongs to.
required	false
type	String
default	`default`

YML

```yml
namespace: production
```

JSON

```json
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the entity or last updated the

entity. Sensu automatically populates the `created_by` field when the entity is created or updated.

required	false
type	String

example	**YML**
	
	``yml
	created_by: admin
	``
	
	JSON
	
	``json
	{
	  "created_by": "admin"
	}
	``

labels

description	Custom attributes to include with observation data in events that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels. *NOTE: For labels that you define in agent.yml or backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the label `proxyType: "website"` in agent.yml or backend.yml, it will be listed as `proxytype: "website"` in entity definitions.* *Key cases are **not** modified for labels you define with a command line flag or an environment variable.*

required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example	**YML**
	
	``yml
	labels:
	  environment: development
	  region: us-west-2
	``
	
	JSON
	
	``json
	{
	  "labels": {
	    "environment": "development",
	    "region": "us-west-2"
	  }
	}
	``

annotations

description	Non-identifying metadata to include with observation data in events that you can access with <u>event filters</u>. 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 <u>API response filtering</u>, <u>sensuctl response filtering</u>, or <u>web UI views</u>. *NOTE: For annotations defined in agent.yml or backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the annotation `webhookURL: "https://my-webhook.com"` in agent.yml or backend.yml, it will be listed as `webhookurl: "https://my-webhook.com"` in entity definitions.* *Key cases are **not** modified for annotations you define with a*

command line flag or an environment variable.

required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

YML

example	

```yaml
annotations:
  managed-by: ops
  playbook: www.example.url
```

JSON

```json
{
  "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

YML

example

```
entity_class: agent
```

JSON

```json
{
  "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.
required	false
type	Array
default	The entity-specific subscription.

YML

example

```yaml
subscriptions:
- web
- prod
- entity:example-entity
```

JSON

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

system

description	System information about the entity, such as operating system and platform. Read system attributes for more information.

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

required	false
type	Map
	YML
example	

```yaml
system:
  arch: amd64
  libc_type: glibc
  vm_system: kvm
  vm_role: host
  cloud_provider: null
  processes:
  - name: Slack
    pid: 1349
    ppid: 0
    status: Ss
    background: true
    running: true
    created: 1582137786
    memory_percent: 1.09932518
    cpu_percent: 0.3263987595984941
  - name: Slack Helper
    pid: 1360
    ppid: 1349
    status: Ss
    background: true
    running: true
    created: 1582137786
    memory_percent: 0.146866455
```

```
      cpu_percent: 0.30897618146109257
  hostname: example-hostname
  network:
    interfaces:
    - addresses:
      - 127.0.0.1/8
      - ::1/128
      name: lo
    - addresses:
      - 93.184.216.34/24
      - 2606:2800:220:1:248:1893:25c8:1946/10
      mac: 52:54:00:20:1b:3c
      name: eth0
  os: linux
  platform: ubuntu
  platform_family: debian
  platform_version: "16.04"
```

JSON

```
{
  "system": {
    "hostname": "example-hostname",
    "os": "linux",
    "platform": "ubuntu",
    "platform_family": "debian",
    "platform_version": "16.04",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "eth0",
          "mac": "52:54:00:20:1b:3c",
          "addresses": [
            "93.184.216.34/24",
            "2606:2800:220:1:248:1893:25c8:1946/10"
```

```
        ]
      }
    ]
  },
  "arch": "amd64",
  "libc_type": "glibc",
  "vm_system": "kvm",
  "vm_role": "host",
  "cloud_provider": "",
  "processes": [
    {
      "name": "Slack",
      "pid": 1349,
      "ppid": 0,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 1.09932518,
      "cpu_percent": 0.3263987595984941
    },
    {
      "name": "Slack Helper",
      "pid": 1360,
      "ppid": 1349,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 0.146866455,
      "cpu_percent": 0.308976181461092553
    }
  ]
  }
}
```

sensu_agent_version

description	Sensu Semantic Versioning (SemVer) version of the agent entity.
required	true
type	String

example

YML

```yml
sensu_agent_version: 1.0.0
```

JSON

```json
{
  "sensu_agent_version": "1.0.0"
}
```

last_seen

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

example

YML

```yml
last_seen: 1522798317
```

JSON

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

deregister

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

```yml
deregister: false
```

JSON

```json
{
  "deregister": false
}
```

deregistration

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

```yml
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

example	

```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

YML

example	

```yaml
  user: agent
```

JSON

```json
{
  "user": "agent"
}
```

System attributes

hostname	
description	Hostname of the entity.
required	false
type	String

example	**YML**
	
	`hostname: example-hostname`
	
	JSON
	
	`{`
	`  "hostname": "example-hostname"`
	`}`

os	
description	Entity's operating system.
required	false

type	String

YML

```
  os: linux
```

JSON

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

platform

description	Entity's operating system distribution.

required	false

type	String

YML

```
  platform: ubuntu
```

JSON

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

platform_family

description	Entity's operating system family.

required	false

type	String

example YML

```
platform_family: debian
```

JSON

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

platform_version

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

example YML

```
platform_version: 16.04
```

JSON

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

network

description	Entity's network interface list. Read network attributes for more information.
required	false

type	Map

YML

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

JSON

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

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	**YML**
	
	```yml\narch: amd64\n```
	
	JSON
	
	```json\n{\n "arch": "amd64"\n}\n```

libc_type

description	Entity's libc type. Automatically populated upon agent startup.
required	false
type	String

example	**YML**
	
	```yml\nlibc_type: glibc\n```
	
	JSON
	
	```json\n{\n "libc_type": "glibc"\n}\n```

vm_system

description	Entity's virtual machine system. Automatically populated upon agent startup.
required	false
type	String

example

YML

```yml
vm_system: kvm
```

JSON

```json
{
  "vm_system": "kvm"
}
```

vm_role

description	Entity's virtual machine role. Automatically populated upon agent startup.
required	false
type	String

example

YML

```yml
vm_role: host
```

JSON

```json
{
  "vm_role": "host"
}
```

cloud_provider

description	Entity's cloud provider environment. Automatically populated upon agent startup if the `--detect-cloud-provider` flag is set. Returned empty unless the agent runs on Amazon Elastic Compute Cloud (EC2), Google Cloud Platform (GCP), or Microsoft Azure. **NOTE**: *This feature can result in several HTTP requests or DNS lookups being performed, so it may not be appropriate for all environments.*
required	false
type	String

example	**YML**
	
	``yml
	"cloud_provider": ""
	``
	
	JSON
	
	``json
	{
	  "cloud_provider": ""
	}
	``

processes

description	List of processes on the local agent. Read processes attributes for more information. **NOTE**: *Process discovery is disabled in this version of Sensu. New events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.*

required	false
type	Map

YML

```yaml
processes:
- name: Slack
  pid: 1349
  ppid: 0
  status: Ss
  background: true
  running: true
  created: 1582137786
  memory_percent: 1.09932518
  cpu_percent: 0.3263987595984941
- name: Slack Helper
  pid: 1360
  ppid: 1349
  status: Ss
  background: true
  running: true
  created: 1582137786
  memory_percent: 0.146866455
  cpu_percent: 0.30897618146109257
```

JSON

```json
{
  "processes": [
    {
      "name": "Slack",
      "pid": 1349,
      "ppid": 0,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 1.09932518,
      "cpu_percent": 0.3263987595984941
    },
    {
```

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

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
| example | ```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	**YML**
	
	``yaml
	name: eth0
	``
	
	JSON
	
	``json
	{
	  "name": "eth0"
	``

```
    }
```

mac

description	Network interface's MAC address.
required	false
type	string

example

YML

```yml
mac: 52:54:00:20:1b:3c
```

JSON

```json
{
  "mac": "52:54:00:20:1b:3c"
}
```

addresses

description	List of IP addresses for the network interface.
required	false
type	Array

example

YML

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

JSON

```json
{
```

```
      "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	**YML**
	
	```handler: email-handler```
	
	JSON
	
	``{
	  "handler": "email-handler"
	}```

Processes attributes

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

name

description	Name of the process.
required	false
type	String

YML

example

```
  name: Slack
```

JSON

```
  {
    "name": "Slack"
  }
```

pid

description	Process ID of the process.
required	false
type	Integer

YML

example

```
  pid: 1349
```

JSON

```
  {
    "pid": 1349
  }
```

ppid

description	Parent process ID of the process.
required	false
type	Integer

example

YML

```yml
ppid: 0
```

JSON

```json
{
  "ppid": 0
}
```

status

description	Status of the process. Read the Linux `top` manual page for examples.
required	false
type	String

example

YML

```yml
status: Ss
```

JSON

```json
{
  "status": "Ss"
}
```

background

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

YML

example
```yml
background: true
```

JSON

```json
{
  "background": true
}
```

running

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

YML

example
```yml
running: true
```

JSON

```json
{
  "running": true
}
```

created

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

YML

example

```yaml
created: 1586138786
```

JSON

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

memory_percent

description	Percent of memory the process is using. The value is returned as a floating-point number where 0.0 = 0% and 1.0 = 100%. For example, the memory_percent value 0.19932 equals 19.932%. *NOTE: The* `memory_percent` *attribute is supported on Linux and macOS. It is not supported on Windows.*
required	false
type	float

YML

example

```yaml
memory_percent: 0.19932
```

JSON

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

cpu_percent

description	Percent of CPU the process is using. The value is returned as a floating-point number where 0.0 = 0% and 1.0 = 100%. For example, the cpu_percent value 0.12639 equals 12.639%.

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

required	false
type	float

YML

example	

```yaml
cpu_percent: 0.12639
```

JSON

```json
{
  "cpu_percent": 0.12639
}
```

Monitor external resources with proxy entities

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

> **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 dynamic runtime assets

To power the check, you'll use the Sensu Plugins HTTP dynamic runtime asset. The Sensu Plugins HTTP asset includes `check-http.rb`, which your check will rely on.

The Sensu assets packaged from Sensu Plugins HTTP are built against the Sensu Ruby runtime environment, so you also need to add the Sensu Ruby Runtime dynamic runtime asset. Sensu Ruby Runtime delivers the Ruby executable and supporting libraries the check will need to run the `check-http.rb` plugin.

Use `sensuctl asset add` to register the Sensu Plugins HTTP dynamic runtime asset, `sensu-plugins/sensu-plugins-http`:

```
sensuctl asset add sensu-plugins/sensu-plugins-http:6.0.0 -r sensu-plugins-http
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu-plugins/sensu-plugins-http:6.0.0
added asset: sensu-plugins/sensu-plugins-http:6.0.0

You have successfully added the Sensu asset resource, but the asset will not get downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the appropriate
resource, populate the "runtime_assets" field with ["sensu-plugins-http"].
```

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

You can also download the dynamic runtime asset definition from Bonsai and register the asset with `sensuctl create --file filename.yml` or `sensuctl create --file filename.json`.

Then, use the following sensuctl example to register the Sensu Ruby Runtime dynamic runtime asset, `sensu/sensu-ruby-runtime`:

```
sensuctl asset add sensu/sensu-ruby-runtime:0.1.0 -r sensu-ruby-runtime
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-ruby-runtime:0.1.0
added asset: sensu/sensu-ruby-runtime:0.1.0

You have successfully added the Sensu asset resource, but the asset will not get downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the appropriate
resource, populate the "runtime_assets" field with ["sensu-ruby-runtime"].
```

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

Use sensuctl to confirm that both dynamic runtime assets are ready to use:

```
sensuctl asset list
```

The response should list the `sensu-plugins-http` and `sensu-ruby-runtime` dynamic runtime assets:

```
        Name                     URL                              Hash
  _____    _____
  _____
  sensu-plugins-http      //assets.bonsai.sensu.io/.../sensu-plugins-http_6.0.0_centos_linux_amd64.tar.gz      31023af
  sensu-ruby-runtime       //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-2.4.4_centos_linux_amd64.tar.gz
  338b88b
```

NOTE: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read *the asset reference* for more information about dynamic runtime asset builds.

Create the check

Now that the dynamic runtime 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` dynamic 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 named `check.yml` or `check.json` in your Sensu installation to store the check definition. Copy this this check definition into the `check.yml` or `check.json` file and save it:

YML

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

```yaml
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

```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"
    ]
  }
}
```

Use sensuctl to add `check-sensu-site` to Sensu directly from the `check.yml` or `check.json` file:

SHELL

```
sensuctl create --file check.yml
```

SHELL

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

Use sensuctl to confirm that Sensu added the check:

```
sensuctl check list
```

The response should list `check-sensu-site`:

Name	Command	Interval	Cron	Timeout	TTL	Subscriptions	Handlers	Assets	Hooks	Publish?	Stdin?
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](#), use sensuctl to add the `proxy` subscription to the entity the Sensu agent is observing.

In the following command, Replace `<entity_name>` with the name of the entity on your system. Then, run:

```
sensuctl entity update <entity_name>
```

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

Validate the check

Use sensuctl to confirm that Sensu created `sensu-site`. It might take a few moments for Sensu to execute the check and create the proxy entity.

```
sensuctl entity list
```

The response should list the `sensu-site` proxy entity:

```
   ID       Class   OS       Subscriptions               Last Seen
 _____ _____ _____ _____
 _____
 sensu-centos  agent  linux  proxy,entity:sensu-centos  2019-01-16 21:50:03 +0000 UTC
 sensu-site    proxy          entity:sensu-site          N/A
```

Then, use sensuctl to confirm that Sensu is monitoring `sensu-site` with the `check-sensu-site` check:

```
sensuctl event info sensu-site check-sensu-site
```

The response should list `check-sensu-site` status and history data for the `sensu-site` proxy entity:

```
=== sensu-site - check-sensu-site
Entity:    sensu-site
Check:     check-sensu-site
Output:    CheckHttp OK: 200, 72024 bytes
Status:    0
History:   0,0
Silenced:  false
Timestamp: 2019-01-16 21:51:53 +0000 UTC
```

You can also view the new proxy entity in your Sensu web UI.

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` dynamic 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 named `entities.yml` or `entities.json` in your Sensu installation and add the following entity definitions:

YML

```
---
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
```

JSON

```
{
  "type": "Entity",
```

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

YML

```yaml
---
type: Entity
api_version: core/v2
metadata:
  name: packagecloud-site
  namespace: default
  labels:
    proxy_type: website
    url: https://packagecloud.io
spec:
  entity_class: proxy
```

JSON

```json
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "packagecloud-site",
    "namespace": "default",
    "labels": {
      "proxy_type": "website",
      "url": "https://packagecloud.io"
    }
  },
```

```
    "spec": {
      "entity_class": "proxy"
    }
  }
```

YML

```yaml
---
type: Entity
api_version: core/v2
metadata:
  name: github-site
  namespace: default
  labels:
    proxy_type: website
    url: https://github.com
spec:
  entity_class: proxy
```

JSON

```json
{
  "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 directly from the `entities.yml` or `entities.json` file:

SHELL

```
sensuctl create --file entities.yml
```

SHELL

```
sensuctl create --file entities.json
```

Use sensuctl to confirm that the entities were added:

```
sensuctl entity list
```

The response should list the new `sensu-docs`, `packagecloud-site`, and `github-site` proxy entities:

```
      ID          Class   OS      Subscriptions            Last Seen
 _____ _____ _____ _____
 _____

 github-site       proxy                    N/A
 packagecloud-site  proxy                      N/A
 sensu-centos        agent  linux  proxy,entity:sensu-centos  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-http.yml` or `check-http.json` in your Sensu installation and add the following check definition:

YML

```yaml
---
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-http` check to Sensu directly from the `check-http.yml` or `check-http.json` file:

SHELL

```
sensuctl create --file check-http.yml
```

SHELL

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

Use sensuctl to confirm that Sensu created the check:

```
sensuctl check list
```

The response should include the `check-http` check:

Name	Command	Interval	Cron	Timeout	TTL	Subscriptions	Handlers	Assets
Hooks	Publish?	Stdin?						
———————	———————————————————————	——————	————	——————	————	——————————	—————
———————	—————	———	———————————————————————	—————	——————

```
 _____
   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, read the* <u>checks reference</u>.

Validate the check

Before you validate the check, make sure that you've <u>registered the</u> `sensu-plugins-http` <u>and</u> `sensu-ruby-runtime` <u>dynamic runtime assets</u> and <u>added the</u> `proxy` <u>subscription to a Sensu agent</u>.

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
```

The response should list check status data for the `sensu-docs`, `packagecloud-site`, and `github-site` proxy entities:

```
    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

The files you created with check and entity definitions can become part of your <u>monitoring as code</u> repository. Storing your Sensu configurations the same way you would store code means they are portable and repeatable. Monitoring as code makes it possible to move to a more robust deployment without losing what you've started here and reproduce one environment's configuration in another.

Now that you know how to run a proxy check to verify 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](#)

Events

Sensu Observability Pipeline
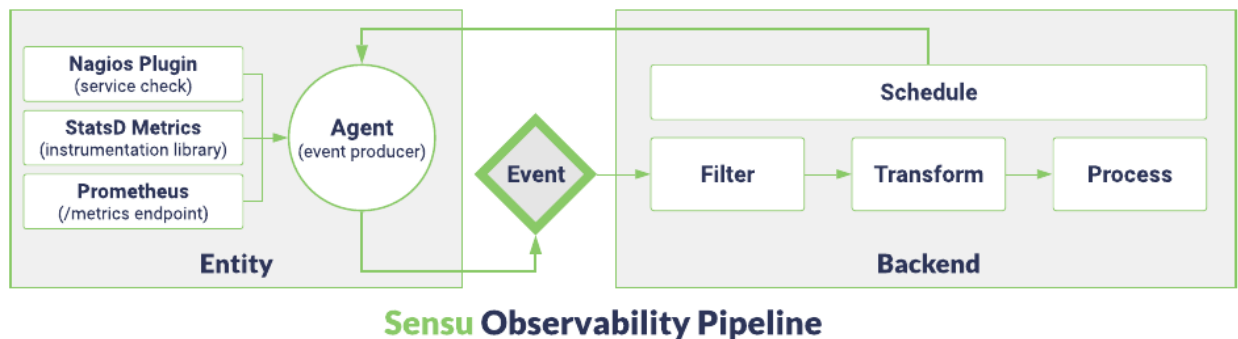
Events are generic containers that Sensu uses to provide context to status and metrics check results. The context, called observation data, is information about the originating entity and the corresponding status or metric check result.

These generic containers allow Sensu to handle different types of events in the pipeline for comprehensive system and service monitoring and observability.Events can contain CPU, memory, and disk usage data; custom application metrics; log data you can send to an external database; and more.

Events require a timestamp, entity, and check.Each event must contain a check result, whether status or metrics.In certain cases, an event can contain both.Because events are polymorphic in nature, it is important to never assume their content (or lack of content).

Here's an example event that includes both status and metrics data:

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"
    duration: 0.060790838
    env_vars: null
```

```yaml
    executed: 1552506033
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1552505833
      status: 0
    - executed: 1552505843
      status: 0
    interval: 10
    is_silenced: false
    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
    last_seen: 1552495139
```

```yaml
  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
    processes: null
  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
  id: 431a0085-96da-4521-863f-c38b480701e9
  sequence: 1
```

JSON

```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,
      "is_silenced": false,
      "issued": 1552506033,
      "last_ok": 1552506033,
      "low_flap_threshold": 0,
```

```json
      "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",
        "processes": null
      },
      "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,
  "id": "431a0085-96da-4521-863f-c38b480701e9",
  "sequence": 1
  }
}
```

Checks

Checks work with the Sensu agent to produce events automatically. You can use checks to monitor server resources, services, and application health as well as collect and analyze metrics.Checks define how Sensu will process events, as well as when and where events are generated via subscriptions and scheduling.

Read Monitor server resources to learn more about using checks to generate events.

Status-only events

A Sensu event is created every time a check result is processed by the Sensu server, regardless of the status the result indicates.The agent creates an event upon receipt of the check execution result and executes any configured hooks the check might have.From there, the status result is forwarded to the Sensu backend, where it is filtered, transformed, and processed.Potentially noteworthy events may be processed by one or more event handlers, for example to send an email or invoke an automated action.

Metrics-only events

Sensu events can be created when the agent receives metrics through the StatsD listener. The agent will translate the StatsD metrics to Sensu metric format and place them inside an event. Because these events do not contain checks, they bypass the store and are sent to the event pipeline and corresponding event handlers.

Status and metrics events

Events that contain *both* a check and metrics most likely originated from check output metric extraction. If a check is configured for metric extraction, the agent will parse the check output and transform it to Sensu metric format. Both the check results and resulting (extracted) metrics are stored inside the event. Event handlers from `event.Check.Handlers` and `event.Metrics.Handlers` will be invoked.

Proxy entities and events

You can create events with proxy entities, which are dynamically created entities that Sensu adds to the entity store if an entity does not already exist for a check result. Proxy entities allow Sensu to monitor external resources on systems where you cannot install a Sensu agent, like a network switch or website. Read Monitor external resources to learn how to use a proxy entity to monitor a website.

Events API

Sensu's events API provides HTTP access to create, retrieve, update, and delete events. If you create a new event that references an entity that does not already exist, the Sensu backend will automatically create a proxy entity when the event is published.

Events reference

An event is a generic container used by Sensu to provide context to checks and metrics. The context, called observation data or event data, contains information about the originating entity and the corresponding check or metric result. An event must contain a status or metrics. In certain cases, an event can contain both a status and metrics. These generic containers allow Sensu to handle different types of events in the pipeline. Because events are polymorphic in nature, it is important to never assume their contents (or lack of content).

Event format

Sensu events contain:

- `entity` scope (required)
    - Information about the source of the event, including any attributes defined in the entity specification
- `check` scope (optional if the `metrics` scope is present)
    - Information about how the event was created, including any attributes defined in the check specification
    - Information about the event and its history, including any check attributes defined in the event specification on this page
- `metrics` scope (optional if the `check` scope is present)
    - Metric points in Sensu metric format
- `timestamp`
    - Time that the event occurred in seconds since the Unix epoch
- `event_id`
    - Universally unique identifier (UUID) for the event

Example status-only event

The following example shows the complete resource definition for a status-only event:

YML

```yaml
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    check_hooks: null
    command: check-cpu-usage -w 75 -c 90
    duration: 5.058211427
    env_vars: null
    executed: 1617050501
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1617050261
      status: 0
    - executed: 1617050321
      status: 0
    - executed: 1617050381
      status: 0
    - executed: 1617050441
      status: 0
    - executed: 1617050501
      status: 0
    interval: 60
    is_silenced: false
    issued: 1617050501
    last_ok: 1617050501
    low_flap_threshold: 0
    metadata:
      name: check_cpu
      namespace: default
    occurrences: 5
    occurrences_watermark: 5
    output: |
      CheckCPU TOTAL OK: total=0.41 user=0.2 nice=0.0 system=0.2 idle=99.59
  iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0
    output_metric_format: ""
```

```yaml
    output_metric_handlers: null
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
    - check-cpu-usage
    scheduler: memory
    secrets: null
    state: passing
    status: 0
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    total_state_change: 0
    ttl: 0
  entity:
    deregister: false
    deregistration: {}
    entity_class: agent
    last_seen: 1617050501
    metadata:
      name: sensu-centos
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
    - api_token
    - access_key
    - secret_key
    - private_key
    - secret
    sensu_agent_version: 6.1.0
    subscriptions:
    - linux
    - entity:sensu-centos
    system:
      arch: amd64
      cloud_provider: ""
```

```
      hostname: sensu-centos
      libc_type: glibc
      network:
        interfaces:
        - addresses:
          - 127.0.0.1/8
          - ::1/128
          name: lo
        - addresses:
          - 10.0.2.15/24
          - fe80::a268:dcce:3be:1c73/64
          mac: 08:00:27:8b:c9:3f
          name: eth0
        - addresses:
          - 172.28.128.45/24
          - fe80::a00:27ff:feb2:dc46/64
          mac: 08:00:27:b2:dc:46
          name: eth1
      os: linux
      platform: centos
      platform_family: rhel
      platform_version: 7.5.1804
      processes: null
      vm_role: guest
      vm_system: vbox
    user: agent
  id: 3c3e68f6-6db7-40d3-9b84-4d61817ae559
  timestamp: 1617050507
```

JSON

```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "check-cpu-usage -w 75 -c 90",
      "duration": 5.058211427,
```

```json
      "env_vars": null,
      "executed": 1617050501,
      "handlers": [],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1617050261,
          "status": 0
        },
        {
          "executed": 1617050321,
          "status": 0
        },
        {
          "executed": 1617050381,
          "status": 0
        },
        {
          "executed": 1617050441,
          "status": 0
        },
        {
          "executed": 1617050501,
          "status": 0
        }
      ],
      "interval": 60,
      "is_silenced": false,
      "issued": 1617050501,
      "last_ok": 1617050501,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "occurrences": 5,
      "occurrences_watermark": 5,
      "output": "CheckCPU TOTAL OK: total=0.41 user=0.2 nice=0.0 system=0.2
  idle=99.59 iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "output_metric_format": "",
      "output_metric_handlers": null,
      "proxy_entity_name": "",
```

```json
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "scheduler": "memory",
      "secrets": null,
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
    },
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1617050501,
      "metadata": {
        "name": "sensu-centos",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "sensu_agent_version": "6.1.0",
      "subscriptions": [
        "linux",
        "entity:sensu-centos"
```

```json
    ],
    "system": {
      "arch": "amd64",
      "cloud_provider": "",
      "hostname": "sensu-centos",
      "libc_type": "glibc",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              ":1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::a268:dcce:3be:1c73/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          },
          {
            "addresses": [
              "172.28.128.45/24",
              "fe80::a00:27ff:feb2:dc46/64"
            ],
            "mac": "08:00:27:b2:dc:46",
            "name": "eth1"
          }
        ]
      },
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "processes": null,
      "vm_role": "guest",
      "vm_system": "vbox"
    },
    "user": "agent"
```

```
    },
    "id": "3c3e68f6-6db7-40d3-9b84-4d61817ae559",
    "timestamp": 1617050507
  }
}
```

Example status-only event from the Sensu API

Sensu sends events to the backend in `json` format, without the outer-level `spec` wrapper or `type` and `api_version` attributes that are included in the `wrapped-json` format. This is the format that events are in when Sensu sends them to handlers:

```
{
  "check": {
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 5.058211427,
    "executed": 1617050501,
    "history": [
      {
        "status": 0,
        "executed": 1617050261
```

```json
        },
        {
          "status": 0,
          "executed": 1617050321
        },
        {
          "status": 0,
          "executed": 1617050381
        },
        {
          "status": 0,
          "executed": 1617050441
        },
        {
          "status": 0,
          "executed": 1617050501
        }
      ],
      "issued": 1617050501,
      "output": "CheckCPU TOTAL OK: total=0.4 user=0.2 nice=0.0 system=0.2 idle=99.6 iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1617050501,
      "occurrences": 5,
      "occurrences_watermark": 5,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check_cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "sensu-centos",
```

```json
      "os": "linux",
    "platform": "centos",
    "platform_family": "rhel",
    "platform_version": "7.5.1804",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "eth0",
          "mac": "08:00:27:8b:c9:3f",
          "addresses": [
            "10.0.2.15/24",
            "fe80::a268:dcce:3be:1c73/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:b2:dc:46",
          "addresses": [
            "172.28.128.45/24",
            "fe80::a00:27ff:feb2:dc46/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "linux",
    "entity:sensu-centos"
  ],
```

```
    "last_seen": 1617049781,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default"
    },
    "sensu_agent_version": "6.1.0"
  },
  "id": "3c3e68f6-6db7-40d3-9b84-4d61817ae559",
  "metadata": {
    "namespace": "default"
  },
  "timestamp": 1617050507
}
```

Example metrics-only event

This example shows the complete resource definition for a metrics-only event:

YML

```
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  entity:
```

```yaml
  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
    processes: null
  user: agent
metrics:
  handlers:
  - influx-db
```

```yaml
    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
  event_id: 47ea07cd-1e50-4897-9e6d-09cd39ec5180
```

JSON

```json
{
  "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": [
```

```json
          "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",
        "processes": null
      },
      "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,
    "event_id": "47ea07cd-1e50-4897-9e6d-09cd39ec5180"
  }
}
```

Example status and metrics event

The following example resource definition for a status and metrics event contains *both* a check and metrics:

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"
    duration: 0.060790838
    env_vars: null
    executed: 1552506033
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1552505833
      status: 0
    - executed: 1552505843
      status: 0
```

```yaml
    interval: 10
    is_silenced: false
    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
    last_seen: 1552495139
    metadata:
      name: sensu-go-sandbox
      namespace: default
    redact:
    - password
    - passwd
    - pass
    - api_key
```

```yaml
      - 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
        processes: null
      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
  event_id: 431a0085-96da-4521-863f-c38b480701e9
```

JSON

```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,
      "is_silenced": false,
      "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",
```

```json
    "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": {
```

```json
    "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",
    "processes": null
  },
  "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,
    "event_id": "431a0085-96da-4521-863f-c38b480701e9"
  }
}
```

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.

If you use the events API to create a new event referencing an entity that does not already exist, the sensu-backend will automatically create a proxy entity in the same namespace when the event is published.

NOTE: An agent cannot belong to, execute checks in, or create events in more than one namespace.

Manage events

You can manage events using the Sensu web UI, events API, and sensuctl command line tool.

View events

To list all events:

```
sensuctl event list
```

To show event details in the default output format (tabular):

```
sensuctl event info <entity-name> <check-name>
```

> **NOTE**: Metrics data points are not included in events retrieved with `sensuctl event info` — these events include check output text rather than a set of metrics points.

With both the `list` and `info` commands, you can specify an output format using the `--format` flag:

- ￢ `yaml` or `wrapped-json` formats for use with sensuctl create
- ￢ `json` format for use with the events API

SHELL

```
sensuctl event info entity-name check-name --format yaml
```

SHELL

```
sensuctl event info entity-name check-name --format wrapped-json
```

SHELL

```
sensuctl event info entity-name check-name --format json
```

Delete events

To delete an event:

```
sensuctl event delete entity-name check-name
```

You can use the `--skip-confirm` flag to skip the confirmation step:

```
sensuctl event delete entity-name check-name --skip-confirm
```

You should receive a confirmation message upon success:

```
Deleted
```

Resolve events

You can use sensuctl to change the status of an event to `0` (OK). Events resolved by sensuctl include the output message `Resolved manually by sensuctl`.

```
sensuctl event resolve entity-name check-name
```

You should receive a confirmation message upon success:

```
Resolved
```

Use event data

Observability data in events is a powerful tool for automating monitoring workflows. For example, the

`state` attribute provides handlers with a high-level description of check status. Filtering events based on this attribute can help reduce alert fatigue.

State attribute

The `state` event attribute adds meaning to the check status:

- `passing` means the check status is `0` (OK).

- `failing` means the check status is non-zero (WARNING or CRITICAL).

- `flapping` indicates an unsteady state in which the check result status (determined based on per-check low and high flap thresholds attributes) is not settling on `passing` or `failing` according to the flap detection algorithm.

Flapping typically indicates intermittent problems with an entity, provided your low and high flap threshold settings are properly configured. Although some teams choose to filter out flapping events to reduce unactionable alerts, we suggest sending flapping events to a designated handler for later review. If you repeatedly observe events in flapping state, Sensu's per-check flap threshold configuration allows you to adjust the sensitivity of the flap detection algorithm.

Flap detection algorithm

Sensu uses the same flap detection algorithm as Nagios. Every time you run a check, Sensu records whether the `status` value changed since the previous check. Sensu stores the last 21 `status` values and uses them to calculate the percent state change for the entity/check pair. Then, Sensu's algorithm applies a weight to these status changes: more recent changes have more value than older changes.

After calculating the weighted total percent state change, Sensu compares it with the low and high flap thresholds set in the check attributes.

- If the entity was **not** already flapping and the weighted total percent state change for the entity/check pair is greater than or equal to the `high_flap_threshold` setting, the entity has started flapping.

- If the entity **was** already flapping and the weighted total percent state change for the entity/check pair is less than the `low_flap_threshold` setting, the entity has stopped flapping.

Depending on the result of this comparison, Sensu will trigger the appropriate event filters based on check attributes like `event.check.high_flap_threshold` and `event.check.low_flap_threshold`.

Occurrences and occurrences watermark

The `occurrences` and `occurrences_watermark` event attributes give you context about recent events for a given entity and check. You can use these attributes within event filters to fine-tune incident notifications and reduce alert fatigue.

Starting at `1`, the `occurrences` attribute increments for events with the same status as the preceding event (OK, WARNING, CRITICAL, or UNKNOWN) and resets whenever the status changes. You can use the `occurrences` attribute to create a state-change-only filter or an interval filter.

The `occurrences_watermark` attribute gives you useful information when looking at events that change status between non-OK (WARNING, CRITICAL, or UNKNOWN) and OK. For these resolution events, the `occurrences_watermark` attribute tells you the number of preceding events with a non-OK status. Sensu resets `occurrences_watermark` to `1` on the first non-OK event. Within a sequence of only OK or only non-OK events, Sensu increments `occurrences_watermark` when the `occurrences` attribute is greater than the preceding `occurrences_watermark`.

The following table shows the occurrences attributes for a series of example events:

event sequence	occurrences	occurrences_watermark
1. OK event	occurrences: 1	occurrences_watermark: 1
2. OK event	occurrences: 2	occurrences_watermark: 2
3. WARNING event	occurrences: 1	occurrences_watermark: 1
4. WARNING event	occurrences: 2	occurrences_watermark: 2
5. WARNING event	occurrences: 3	occurrences_watermark: 3
6. CRITICAL event	occurrences: 1	occurrences_watermark: 3
7. CRITICAL event	occurrences: 2	occurrences_watermark: 3
8. CRITICAL event	occurrences: 3	occurrences_watermark: 3
9. CRITICAL event	occurrences: 4	occurrences_watermark: 4
10. OK event	occurrences: 1	occurrences_watermark: 4
11. CRITICAL event	occurrences: 1	occurrences_watermark: 1

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 | **YML**<br><br>```yaml
type: Event
```<br><br>**JSON**<br><br>```json
{
  "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	**YML**

```
api_version: core/v2
```

JSON

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

metadata	
description	Top-level scope that contains the event `namespace` and `created_by` field. The `metadata` map is always at the top level of the check definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Review the metadata attributes for details.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML**

```yaml
metadata:
  namespace: default
  created_by: admin
```

JSON

```json
{
  "metadata": {
    "namespace": "default",
    "created_by": "admin"
  }
}
```

spec	
description	Top-level map that includes the event spec attributes.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs **YML**
example	

```yaml
spec:
  check:
    check_hooks:
    command: /opt/sensu-plugins-ruby/embedded/bin/metrics-
curl.rb -u "http://localhost"
    duration: 0.060790838
    env_vars:
    executed: 1552506033
    handlers: []
    high_flap_threshold: 0
    history:
    - executed: 1552505833
      status: 0
    - executed: 1552505843
      status: 0
    interval: 10
    is_silenced: true
    issued: 1552506033
    last_ok: 1552506033
    low_flap_threshold: 0
    metadata:
      name: curl_timings
      namespace: default
    occurrences: 1
    occurrences_watermark: 1
    silenced:
    - webserver:*
    output: |-
      sensu-go-sandbox.curl_timings.time_total 0.005
1552506033
        sensu-go-sandbox.curl_timings.time_namelookup 0.004
    output_metric_format: graphite_plaintext
```

```yaml
      output_metric_handlers:
      - influx-db
      proxy_entity_name: ''
      publish: true
      round_robin: false
      runtime_assets: []
      state: passing
      status: 0
      stdin: false
      subdue:
      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
        processes:
      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
    event_id: 431a0085-96da-4521-863f-c38b480701e9
```

JSON

```
  {
    "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,
      "is_silenced": true,
      "issued": 1552506033,
      "last_ok": 1552506033,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "curl_timings",
        "namespace": "default"
      },
      "occurrences": 1,
      "occurrences_watermark": 1,
      "silenced": [
        "webserver:*"
      ],
      "output": "sensu-go-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,
```

```json
      "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"
```

```json
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          }
        ]
      },
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "processes": null
    },
    "user": "agent"
  },
  "metrics": {
    "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,
  "event_id": "431a0085-96da-4521-863f-c38b480701e9"
  }
}
```

Metadata attributes

namespace	
description	Sensu RBAC namespace that the event belongs to.
required	false
type	String
default	`default`

example	**YML**
	``yml
	namespace: production
	``
	
	JSON
	``json
	{
	  "namespace": "production"
	}
	``

created_by	
description	Username of the Sensu user who created the event or last updated the event. Sensu automatically populates the `created_by` field when the event is created or updated.
required	false
type	String

example	**YML**
	``yml
	created_by: "admin"
	``
	
	JSON

```
{
  "created_by": "admin"
}
```

Spec attributes

timestamp	
description	Time that the event occurred. In seconds since the Unix epoch.
	Sensu automatically populates the timestamp value for the event. For events created via the events API, you can specify a `timestamp` value in the request body.
required	false
type	Integer
default	Time that the event occurred
example	**YML**

```
timestamp: 1522099512
```

JSON

```
{
  "timestamp": 1522099512
}
```

event_id	
description	Universally unique identifier (UUID) for the event.
	Sensu automatically populates the `event_id` value for the event.

required	false
type	String

YML

```yml
event_id: 431a0085-96da-4521-863f-c38b480701e9
```

JSON

```json
{
  "event_id": "431a0085-96da-4521-863f-c38b480701e9"
}
```

entity

description	Entity attributes from the originating entity (agent or proxy). For events created with the events API, if the event's entity does not already exist, the sensu-backend automatically creates a proxy entity when the event is published.
type	Map
required	true

YML

```yml
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
```

JSON

```
{
  "entity": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "agent",
    "last_seen": 1552495139,
    "metadata": {
      "name": "sensu-go-sandbox",
      "namespace": "default"
    },
    "redact": [
      "password",
```

```json
          "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	

```yml
check:
  check_hooks:
  command: /opt/sensu-plugins-ruby/embedded/bin/metrics-
curl.rb -u "http://localhost"
  duration: 0.060790838
  env_vars:
  executed: 1552506033
  handlers: []
  high_flap_threshold: 0
  history:
  - executed: 1552505833
    status: 0
  - executed: 1552505843
    status: 0
  interval: 10
  is_silenced: true
  issued: 1552506033
  last_ok: 1552506033
  low_flap_threshold: 0
  metadata:
    name: curl_timings
    namespace: default
  occurrences: 1
  occurrences_watermark: 1
  silenced:
  - webserver:*
  output: sensu-go-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:
    subscriptions:
    - entity:sensu-go-sandbox
    timeout: 0
    total_state_change: 0
    ttl: 0
```

JSON

```
{
  "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,
    "is_silenced": true,
    "issued": 1552506033,
```

```
      "last_ok": 1552506033,
      "low_flap_threshold": 0,
      "metadata": {
        "name": "curl_timings",
        "namespace": "default"
      },
      "occurrences": 1,
      "occurrences_watermark": 1,
      "silenced": [
        "webserver:*"
      ],
      "output": "sensu-go-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. Review the metrics attributes.

type	Map
required	false
	YML
| example | ```yaml
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
``` |

JSON

```json
{
  "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
      }
    ]
  }
```

YML

```
  }
```

Check attributes

Sensu events include a `check` scope that contains information about how the event was created, including any attributes defined in the check specification, and information about the event and its history, including the attributes defined below.

duration	
description	Command execution time. In seconds.
required	false
type	Float

YML

```
  duration: 1.903135228
```

JSON

```
  {
    "duration": 1.903135228
  }
```

executed	
description	Time at which the check request was executed. In seconds since the Unix epoch. The difference between a request's `issued` and `executed` values is the request latency. For agent-executed checks, Sensu automatically populates the

`executed` value. For events created with the events API, the default `executed` value is `0` unless you specify a value in the request body.

required	false
type	Integer

YML

example

```yml
executed: 1522100915
```

JSON

```json
{
  "executed": 1522100915
}
```

history

description	Check status history for the last 21 check executions. Read history attributes. Sensu automatically populates the history attributes with check execution data. To store more than the last 21 check executions, use one of our long-term event storage integrations.
required	false
type	Array

YML

example

```yml
history:
- executed: 1552505983
  status: 0
- executed: 1552505993
  status: 0
```

JSON

```json
{
  "history": [
    {
      "executed": 1552505983,
      "status": 0
    },
    {
      "executed": 1552505993,
      "status": 0
    }
  ]
}
```

issued

description	Time that the check request was issued. In seconds since the Unix epoch.
	The difference between a request's `issued` and `executed` values is the request latency.
	For agent-executed checks, Sensu automatically populates the `issued` value. For events created with the events API, the default `issued` value is `0` unless you specify a value in the request body.
required	false
type	Integer
example	**YML**

```yml
issued: 1552506033
```

JSON

```json
{
  "issued": 1552506033
}
```

last_ok

description	Last time that the check returned an OK status (`0`). In seconds since the Unix epoch. For agent-executed checks, Sensu automatically populates the `last_ok` value. For events created with the events API, the `last_ok` attribute will default to `0` even if you specify OK status in the request body.
required	false
type	Integer

YML

```
last_ok: 1552506033
```

JSON

```
{
  "last_ok": 1552506033
}
```

occurrences

description	Number of preceding events with the same status as the current event (OK, WARNING, CRITICAL, or UNKNOWN). Starting at `1` , the `occurrences` attribute increments for events with the same status as the preceding event and resets whenever the status changes. Read Use event data for more information. Sensu automatically populates the `occurrences` value. For events created with the events API, Sensu overwrites any `occurences` value you specify in the request body with the correct value.

required	false
type	Integer greater than 0

YML

```yml
occurrences: 1
```

JSON

```json
{
  "occurrences": 1
}
```

occurrences_watermark

description	For incident and resolution events, the number of preceding events with an OK status (for incident events) or non-OK status (for resolution events). The `occurrences_watermark` attribute gives you useful information when looking at events that change status between OK (`0`)and non-OK (`1` -WARNING, `2` -CRITICAL, or UNKNOWN).
	Sensu resets `occurrences_watermark` to `1` whenever an event for a given entity and check transitions between OK and non-OK. Within a sequence of only OK or only non-OK events, Sensu increments `occurrences_watermark` only when the `occurrences` attribute is greater than the preceding `occurrences_watermark`. Read Use event data for more information.
	Sensu automatically populates the `occurrences_watermark` value. In events created with the events API, Sensu overwrites any `occurences_watermark` value you specify in the request body with the correct value.
required	false
type	Integer greater than 0

YML

example	

```
occurrences_watermark: 1
```

JSON

```
{
  "occurrences_watermark": 1
}
```

is_silenced

description	If `true`, the event was silenced at the time of processing. Otherwise, `false`. If `true`, the event. Check definitions also list the silenced entries that match the event in the `silenced` array.
required	false
type	Boolean

YML

example	

```
is_silenced: true
```

JSON

```
{
  "is_silenced": "true"
}
```

silenced

description	Array of silencing entries that match the event. The `silenced` attribute is only present for events if one or more silencing entries matched the event at time of processing. If the `silenced` attribute is not present in an event, the event was not silenced at the time of processing.

required	false
type	Array

YML

```yaml
silenced:
- webserver:*
```

JSON

```json
{
  "silenced": [
    "webserver:*"
  ]
}
```

output	
description	Output from the execution of the check command.
required	false
type	String

YML

```yaml
output: "sensu-go-sandbox.curl_timings.time_total 0.005
```

JSON

```json
{
  "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`. Use the `low_flap_threshold` and `high_flap_threshold` check attributes to configure `flapping` state detection. Sensu automatically populates the `state` based on the `status`.
required	false
type	String

example	**YML**
	
	 ``yaml
	state: passing
	``
	
	 JSON
	
	 ``json
	{
	  "state": "passing"
	}
	``

status

description	Exit status code produced by the check. ¬ `0` indicates OK ¬ `1` indicates WARNING ¬ `2` indicates CRITICAL Exit status codes other than `0`, `1`, or `2` indicate an UNKNOWN or custom status.. For agent-executed checks, Sensu automatically populates the `status` value based on the check result. For events created with the events API, Sensu assumes the status is `0` (OK) unless you specify a non-zero value in the request body.

required	false
type	Integer

	YML
example	

```yml
status: 0
```

JSON

```json
{
  "status": 0
}
```

total_state_change

description	Total state change percentage for the check's history.
	For agent-executed checks, Sensu automatically populates the `total_state_change` value. For events created with the events API, the `total_state_change` attribute will default to `0` even if you specify a different value or change the `status` value in the request body.

required	false
type	Integer

	YML
example	

```yml
total_state_change: 0
```

JSON

```json
{
  "total_state_change": 0
}
```

History attributes

executed	
description	Time at which the check request was executed. In seconds since the Unix epoch.
	Sensu automatically populates the `executed` value with check execution data. For events created with the events API, the `executed` default value is `0`.
required	false
type	Integer

YML

example
```
executed: 1522100915
```

JSON

```
{
  "executed": 1522100915
}
```

status	
description	Exit status code produced by the check.

- ¬ `0` indicates OK
- ¬ `1` indicates WARNING
- ¬ `2` indicates CRITICAL

Exit status codes other than `0`, `1`, or `2` indicate an UNKNOWN or custom status.

Sensu automatically populates the `status` value with check execution data.

required	false
type	Integer

YML

example	

```yaml
status: 0
```

JSON

```json
{
  "status": 0
}
```

Metrics attributes

handlers	
description	Array of Sensu handlers to use for events created by the check. Each array item must be a string.
required	false
type	Array

YML

example	

```yaml
handlers:
- influx-db
```

JSON

```json
{
  "handlers": [
    "influx-db"
  ]
}
```

points

description	Metrics data points, including a name, timestamp, value, and tags. Read points attributes.
required	false
type	Array

YML

example

```yaml
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
```

JSON

```json
{
  "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	**YML**

```
name: sensu-go-sandbox.curl_timings.time_total
```

JSON

```
{
  "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	**YML**

```yaml
tags:
- name: response_time_in_ms
  value: '101'
```

JSON

```json
{
  "tags": [
    {
      "name": "response_time_in_ms",
      "value": "101"
    }
  ]
}
```

timestamp

description	Time at which the metric was collected. In seconds since the Unix epoch. Sensu automatically populates the timestamp values for metrics data points.
required	false
type	Integer

example	**YML**

```
  timestamp: 1552506033
```

JSON

```json
{
  "timestamp": 1552506033
}
```

value

description	Metric value.
required	false
type	Float

YML

example

```yaml
  value: 0.005
```

JSON

```json
{
  "value": 0.005
}
```

Schedule observability data collection

Sensu Observability Pipeline
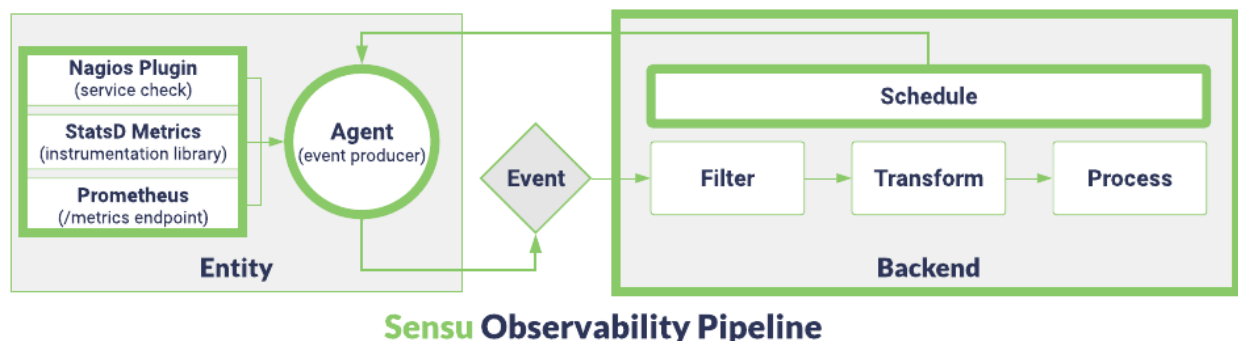
Sensu's schedule function is based on <u>subscriptions</u>: transport topics to which the Sensu <u>backend</u> publishes check requests. The subscriptions you specify in your Sensu <u>agent</u> definition determine which <u>checks</u> the agent will execute. The Sensu backend schedules checks, publishes check execution requests to <u>entities</u>, and processes the <u>observation data (events)</u> it receives from the agent.

Agent and backend

The Sensu agent is a lightweight process that runs on the infrastructure components you want to monitor and observe. The agent registers with the Sensu backend as an entity with `type: "agent"`. Agent entities are responsible for creating <u>status and metrics events</u> to send to the <u>backend event pipeline</u>.

The Sensu backend includes an integrated structure for scheduling checks using subscriptions and an event pipeline that applies <u>event filters</u>, <u>mutators</u>, and <u>handlers</u>, an embedded <u>etcd</u> datastore for storing configuration and state, and the Sensu <u>API</u>, Sensu <u>web UI</u>, and <u>sensuctl</u> command line tool.

The Sensu agent is available for Linux, macOS, and Windows. The Sensu backend is available for Ubuntu/Debian and RHEL/CentOS distributions of Linux. Learn more in the <u>agent</u> and <u>backend</u> references.

Follow the <u>installation guide</u> to install the agent and backend.

Subscriptions

Subscriptions are at the core of Sensu's publish/subscribe pattern of communication: subscriptions are transport topics to which the Sensu backend publishes check requests.Sensu entities become subscribers to these topics via their individual `subscriptions` attribute.

Each Sensu agent's defined set of subscriptions determine which checks the agent will execute.Agent subscriptions allow Sensu to request check executions on a group of systems at a time instead of a traditional 1:1 mapping of configured hosts to monitoring checks.

In each check's definition, you can specify which subscriptions should run the check.At the same time, your entities are "subscribed" to these subscriptions.Subscriptions make sure your entities automatically run the appropriate checks for their functionality.

The following example shows the resource definition for a check with the `system` and `linux` subscriptions.This check would run on any entities whose definitions also specify the `system` or `linux` subscriptions.

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: check-cpu
  namespace: default
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers:
  - slack
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
```

```
    subdue: null
    subscriptions:
    - system
    - linux
    timeout: 0
    ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "check-cpu",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
```

```
        "timeout": 0,
        "ttl": 0
    }
  }
```

Subscriptions typically correspond to a specific role or responsibility. For example, you might add all the checks you want to run on your database entities to a `database` subscription. Rather than specifying these checks individually for every database you are monitoring, you add the `database` subscription to your database entities and they run the desired checks automatically.

Read the subscriptions reference to learn more.

Communication between the agent and backend

The Sensu agent uses WebSocket (ws) protocol to send and receive JSON messages with the Sensu backend. For optimal network throughput, agents will attempt to negotiate the use of Protobuf serialization when communicating with a Sensu backend that supports it. This communication is via clear text by default.

Follow Secure Sensu to configure the backend and agent for WebSocket Secure (wss) encrypted communication.

Agent reference

Example Sensu agent configuration file (download)

The Sensu agent is a lightweight client that runs on the infrastructure components you want to monitor. Agents register with the Sensu backend as 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. For Windows operating systems, the Sensu agent uses `cmd.exe` for the execution environment. For all other operating systems, the Sensu agent uses the Bourne shell (sh).

Read the installation guide to install the agent.

Agent authentication

The Sensu agent authenticates to the Sensu backend via WebSocket transport by either built-in basic (username and password) or mutual transport layer security (mTLS) authentication.

Username and password authentication

The default mechanism for agent authentication is built-in basic authentication with username and password. The Sensu agent uses username and password authentication unless mTLS authentication has been explicitly configured.

For username and password authentication, sensu-agent joins the username and password with a colon and encodes them as a Base64 value. Sensu provides the encoded string as the value of the `Authorization` HTTP header — for example, `Authorization: Basic YWdlbnQ6UEBzc3cwcmQh` — to authenticate to the Sensu backend.

When using username and password authentication, sensu-agent also sends the following HTTP headers in requests to the backend:

- `Sensu-User` : the username in plaintext

- `Sensu-AgentName` : the agent's configured name in plaintext

- `Sensu-Subscriptions` : the agent's subscriptions in a comma-separated plaintext list
- `Sensu-Namespace` : the agent's configured namespace in plaintext

mTLS authentication

When mTLS is configured for both the Sensu agent and backend, the agent uses mTLS authentication instead of the default username and password authentication.

Sensu backends that are configured for mTLS authentication will no longer accept agent authentication via username and password. Agents that are configured to use mTLS authentication cannot authenticate with the backend unless the backend is configured for mTLS.

To configure the agent and backend for mTLS authentication:

- In the backend configuration, specify valid certificate and key files as values for the `agent-auth-cert-file` and `agent-auth-key-file` parameters (e.g. `backend-1.pem` and `backend-1-key.pem` , respectively).
- In the agent configuration, specify valid certificate and key files as values for the `cert-file` and `key-file` parameters (e.g. `agent.pem` and `agent-key.pem` , respectively).

The agent and backend will compare the provided certificates with the trusted CA certificate either in the system trust store or specified explicitly as the `agent-auth-trusted-ca-file` in the backend configuration and `trusted-ca-file` in the agent configuration.

When using mTLS authentication, sensu-agent sends the following HTTP headers in requests to the backend:

- `Sensu-AgentName` : the agent's configured name in plaintext
- `Sensu-Subscriptions` : the agent's subscriptions in a comma-separated, plaintext list
- `Sensu-Namespace` : the agent's configured namespace in plaintext

If the Sensu agent is configured for mTLS authentication, it will not send the `Authorization` HTTP header.

Certificate revocation check

The Sensu backend checks certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP) endpoints for agent mTLS, etcd client, and etcd peer connections whose remote sides present X.509 certificates that provide CRL and OCSP revocation information.

Communication between the agent and backend

The Sensu agent uses WebSocket (ws) protocol to send and receive JSON messages with the Sensu backend. For optimal network throughput, agents will attempt to negotiate the use of Protobuf serialization when communicating with a Sensu backend that supports it. This communication is via clear text by default.

Follow Secure Sensu to configure the backend and agent for WebSocket Secure (wss) encrypted communication.

Create observability 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 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. 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 dynamic runtime assets and stores them in its local cache. By default, agents cache dynamic runtime 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.

Read the subscriptions reference for more information.

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 observation data in events in place of the agent entity data. read the entity reference and Monitor external resources for more information about monitoring proxy entities.

Create observability events using the agent API

The Sensu agent API allows external sources to send monitoring data to Sensu without requiring the external sources to know anything about Sensu's internal implementation. The agent API listens on the address and port specified 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 observability 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. Read 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.

The event will be handled according to an `email` handler definition.

> **NOTE**: For HTTP `POST` requests to the agent /events API, check spec attributes are not required. When doing so, the spec attributes (including `handlers`) are listed as individual top-level attributes in the check definition instead.

```
curl -X POST \
```

```
  -H 'Content-Type: application/json' \
  -d '{
    "check": {
      "metadata": {
        "name": "check-mysql-status"
      },
      "handlers": ["email"],
      "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, read the checks 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 observability events, this approach allows you to worry less about handling every possible error case.A lack of additional events before the 7-hour period elapses results in an alert.

If your backup script runs successfully, 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	<pre>{
	  "check": {
	    "metadata": {
	      "name": "check-mysql-status"
	    },
	    "status": 1,
	    "output": "could not connect to mysql"
	  }
	}</pre>

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 observability events using the StatsD listener

Sensu agents include a listener to send StatsD metrics to the event pipeline.By default, Sensu agents listen on UDP socket 8125 for messages that follow the StatsD line protocol and send metric events for handling by the Sensu backend.

For example, you can use the Netcat utility to send metrics to the StatsD listener:

```
echo 'abc.def.g:10|c' | nc -w1 -u localhost 8125
```

Sensu does not store metrics received through the StatsD listener, so it's important to configure event handlers.

StatsD line protocol

The Sensu StatsD listener accepts messages formatted according to the StatsD line protocol:

```
<metricname>:<value>|<type>
```

For more information, read 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.For example, to start an agent that sends StatsD metrics to InfluxDB, run:

```
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.For example, to start an agent with a customized address and flush interval, run:

```
sensu-agent --statsd-event-handlers influx-db --statsd-flush-interval 1 --statsd-
metrics-host 123.4.5.11 --statsd-metrics-port 8125
```

Create observability events using the agent TCP and UDP sockets

> **NOTE**: The agent TCP and UDP sockets are deprecated in favor of the _agent API_.

Sensu agents listen for external monitoring data using TCP and UDP sockets. The agent sockets accept JSON event data and pass events to the Sensu backend event pipeline for processing. The TCP and UDP sockets listen on the address and port specified by the socket configuration 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	<pre>{
	  "name": "check-mysql-status"
	}</pre>

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	<pre>{
	  "status": 0
	}</pre>

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	Name of the Sensu entity associated with the event. Use this attribute to tie the event to a proxy entity. If no matching entity exists, Sensu creates a proxy entity with the name provided by the `client` attribute. *NOTE: The `client` attribute is deprecated in favor of the `source` attribute.*
required	false
default	The agent entity that receives the event data.
type	String
example	

```
{
  "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	<pre>{
	  "executed": 1458934742
	}</pre>

duration

description	Amount of time it took to execute the check. In seconds.
required	false
type	Float

example	<pre>{
	  "duration": 1.903135228
	}</pre>

command

description	Command executed to produce the event. Use the `command` attribute to add context to the event data. Sensu does not execute the command

included in this attribute.

required	false
type	String
| example | ```json
{
  "command": "http-check --url https://sensu.io"
}
``` |

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 | ```json
{
  "interval": 60
}
``` |

handlers

description	Array of Sensu handler names to use for handling the event. Each handler name in the array must be a string.
required	false
type	Array
| example | ```
{
``` |

```
    "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 web UI.The `keepalive-critical-timeout` is set to `0` (disabled) by default to help ensure that it will not interfere with your `keepalive-warning-timeout` setting.

If a Sensu agent fails to send keepalive events over the period specified by the `keepalive-warning-timeout` flag, the Sensu backend creates a keepalive **warning** alert in the Sensu web UI.The value you specify for `keepalive-warning-timeout` must be lower than the value you specify for `keepalive-critical-timeout` .

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

You can use keepalives to identify unhealthy systems and network partitions, send notifications, and trigger auto-remediation, among other useful actions.In addition, the agent maps `keepalive-critical-timeout` and `keepalive-warning-timeout` values to certain event check attributes, so you can create time-based event filters to reduce alert fatigue for agent keepliave events.

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

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

```
---
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"
    ]
  }
}
```

You can also use the `keepalive-handlers` flag to send keepalive events to any handler you have

configured. If you do not specify a keepalive handler with the `keepalive-handlers` flag, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI.

Connection failure

Although connection failure may be due to different kinds of socket errors (such as unexpectedly closed connections and TLS handshake failures), the Sensu agent generally keeps retrying connections to each URL in the `backend-url` list until it is successfully connected to a backend URL or you stop the process.

When you start up a Sensu agent configured with multiple `backend-url` values, the agent shuffles the `backend-url` list and attempts to connect to the first URL in the shuffled list.

If the agent cannot establish a WebSocket connection with the first URL within the number of seconds specified for the `backend-handshake-timeout`, the agent abandons the connection attempt and tries the next URL in the shuffled list.

When the agent establishes a WebSocket connection with a backend URL within the `backend-handshake-timeout` period, the agent sends a heartbeat message to the backend at the specified `backend-heartbeat-interval`. For every heartbeat the agent sends, the agent expects the connected backend to send a heartbeat response within the number of seconds specified for the `backend-heartbeat-timeout`. If the connected backend does not respond within the `backend-heartbeat-timeout` period, the agent closes the connection and attempts to connect to the next backend URL in the shuffled list.

The agent iterates through the shuffled `backend-url` list until it successfully establishes a WebSocket connection with a backend, returning to the first URL if it fails to connect with the last URL in the list.

> **NOTE**: Sensu's WebSocket connection heartbeat message and _keepalive monitoring_ mechanism are different, although they have similar purposes.
>
> The WebSocket `backend-heartbeat-interval` and `backend-heartbeat-timeout` are specifically configured for the WebSocket connection heartbeat message the agent sends when it connects to a backend URL.
>
> Keepalive monitoring is more fluid — it permits agents to reconnect any number of times within the configured timeout. As long as the agent can successfully send one event to any backend within the timeout, the keepalive logic is satisfied.

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 view 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 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 view the status of the agent service using a service manager:

Linux

```
service sensu-agent status
```

Windows

```
sc.exe query SensuAgent
```

Get service version

There are two ways to get the current agent version: the `sensu-agent` tool and the agent version API.

To get the version of the current `sensu-agent` tool:

```
sensu-agent version
```

To get the version of the running `sensu-agent` service:

```
curl http://127.0.0.1:3031/version
```

Uninstall the service

Windows

```
sensu-agent service uninstall
```

Get help

The `sensu-agent` tool provides general and command-specific help flags.

To view sensu-agent commands, run:

```
sensu-agent help
```

To list options for a specific command (in this case, sensu-agent start), run:

```
sensu-agent start --help
```

Registration, endpoint management, and service discovery

Sensu agents automatically discover and register infrastructure components and the services running on them. When an agent process stops, the Sensu backend can automatically create and process a deregistration event.

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, read 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 via flags

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. Review the example Sensu agent configuration file for flags and defaults.

Certificate bundles or chains

The Sensu agent supports all types of certificate bundles (or chains) as long as the agent (or leaf) certificate is the *first* certificate in the bundle. This is because the Go standard library assumes that the

first certificate listed in the PEM file is the leaf certificate — the certificate that the program will use to show its own identity.

If you send the leaf certificate alone instead of sending the whole bundle with the leaf certificate first, you will receive a `certificate not signed by trusted authority` error. You must present the whole chain to the remote so it can determine whether it trusts the presented certificate through the chain.

Configuration summary

> **NOTE**: *Process discovery is disabled in this version of Sensu. The* `--discover-processes` *flag is not available, and new events will not include data in the* `processes` *attributes. Instead, the field will be empty:* `"processes": null`.

To view configuration information for the sensu-agent start command, run:

```
sensu-agent start --help
```

The response will list command information and configuration flags for sensu-agent start:

```
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)
      --assets-burst-limit int           asset fetch burst limit (default 100)
      --assets-rate-limit float         maximum number of assets fetched per
second
      --backend-handshake-timeout int    number of seconds the agent should wait
```

```
when negotiating a new WebSocket connection (default 15)
      --backend-heartbeat-interval int     interval at which the agent should send
heartbeats to the backend (default 30)
      --backend-heartbeat-timeout int      number of seconds the agent should wait
for a response to a hearbeat (default 45)
      --backend-url strings                comma-delimited list of ws/wss URLs of
Sensu backend servers. This flag can also be invoked multiple times (default
[ws://127.0.0.1:8081])
      --cache-dir string                   path to store cached data (default
"/var/cache/sensu/sensu-agent")
      --cert-file string                   certificate for TLS authentication
  -c, --config-file string                 path to sensu-agent config file (default
"/etc/sensu/agent.yml")
      --deregister                         ephemeral agent
      --deregistration-handler string      deregistration handler that should
process the entity deregistration event
      --detect-cloud-provider              enable cloud provider detection
      --disable-api                        disable the Agent HTTP API
      --disable-assets                     disable check assets on this agent
      --disable-sockets                    disable the Agent TCP and UDP event
sockets
      --discover-processes                 indicates whether process discovery
should be enabled
      --events-burst-limit int             /events api burst limit (default 10)
      --events-rate-limit float            maximum number of events transmitted to
the backend through the /events api
  -h, --help                               help for start
      --insecure-skip-tls-verify           skip TLS verification (not recommended!)
      --keepalive-critical-timeout uint32  number of seconds until agent is
considered dead by backend to create a critical event
      --keepalive-handlers strings         comma-delimited list of keepalive
handlers for this entity. This flag can also be invoked multiple times
      --keepalive-interval int             number of seconds to send between
keepalive events (default 20)
      --keepalive-warning-timeout uint32   number of seconds until agent is
considered dead by backend to create a warning event (default 120)
      --key-file string                    key for TLS authentication
      --labels stringToString              entity labels map (default [])
      --log-level string                   logging level [panic, fatal, error,
warn, info, debug] (default "info")
      --name string                        agent name (defaults to hostname)
(default "my_hostname")
```

```
        --namespace string                       agent namespace (default "default")
        --password string                        agent password (default "P@ssw0rd!")
        --redact strings                         comma-delimited list of fields to redact,
    overwrites the default fields. This flag can also be invoked multiple times (default
    [password,passwd,pass,api_key,api_token,access_key,secret_key,private_key,secret])
        --require-fips                           indicates whether fips support should be
    required in openssl
        --require-openssl                        indicates whether openssl should be
    required instead of go's built-in crypto
        --socket-host string                     address to bind the Sensu client socket
    to (default "127.0.0.1")
        --socket-port int                        port the Sensu client socket listens on
    (default 3030)
        --statsd-disable                         disables the statsd listener and metrics
    server
        --statsd-event-handlers strings      comma-delimited list of event handlers
    for statsd metrics. This flag can also be invoked multiple times
        --statsd-flush-interval int          number of seconds between statsd flush
    (default 10)
        --statsd-metrics-host string             address used for the statsd metrics
    server (default "127.0.0.1")
        --statsd-metrics-port int                port used for the statsd metrics server
    (default 8125)
        --subscriptions strings                  comma-delimited list of agent
    subscriptions. This flag can also be invoked multiple times
        --trusted-ca-file string                 TLS CA certificate bundle in PEM format
        --user string                            agent user (default "agent")
```

Windows

You can specify the agent configuration using a `.yml` file.Review 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

NOTE: *Docker-only Sensu binds to the hostnames of containers, represented here as* `SENSU_HOSTNAME` *in Docker default values.*

allow-list

description	Path to yaml or json file that contains the allow list of check or hook commands the agent can execute. Read allow list configuration commands and the example allow list configuration file for information about building a configuration file.
type	String
default	`""`
environment variable	`SENSU_ALLOW_LIST`
command line example	```sensu-agent start --allow-list /etc/sensu/check-allow-list.yaml```
/etc/sensu/agent.yml example	```allow-list: /etc/sensu/check-allow-list.yaml```

annotations

description	Non-identifying metadata to include with event data that you can access with event filters and tokens. You can use annotations to add data that is meaningful to people or external tools that interact with Sensu.
	In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI view filtering.
	NOTE: For annotations that you define in agent.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the annotation `webhookURL: "https://my-webhook.com"` in agent.yml, it will be listed as `webhookurl: "https://my-webhook.com"` in entity definitions.
	*Key cases are **not** modified for annotations you define with the `--annotations` command line flag or the `SENSU_ANNOTATIONS` environment variable.*

required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_ANNOTATIONS`
| command line example | ```
sensu-agent start --annotations
sensu.io/plugins/slack/config/webhook-
url=https://hooks.slack.com/services/T00000000/B00000000/XX
XXXXXXXXXXXXXXXXXXXXX
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"
``` |

assets-burst-limit

description	Maximum amount of burst allowed in a rate interval when fetching dynamic runtime assets.
type	Integer
default	`100`
environment variable	`SENSU_ASSETS_BURST_LIMIT`
| command line example | ```
sensu-agent start --assets-burst-limit 100
``` |
| /etc/sensu/agent.yml | |

example	assets-burst-limit: 100

assets-rate-limit

description	Maximum number of dynamic runtime assets to fetch per second. The default value `1.39` is equivalent to approximately 5000 user-to-server requests per hour.
type	Float
default	`1.39`
environment variable	`SENSU_ASSETS_RATE_LIMIT`
command line example	`sensu-agent start --assets-rate-limit 1.39`
/etc/sensu/agent.yml example	`assets-rate-limit: 1.39`

backend-handshake-timeout

description	Number of seconds the Sensu agent should wait when negotiating a new WebSocket connection.
type	Integer
default	`15`
environment variable	`SENSU_BACKEND_HANDSHAKE_TIMEOUT`
command line example	`sensu-agent start --backend-handshake-timeout 20`
/etc/sensu/agent.yml example	`backend-handshake-timeout: 20`

backend-heartbeat-interval

description	Interval at which the agent should send heartbeats to the Sensu backend. In seconds.
type	Integer
default	`30`
environment variable	`SENSU_BACKEND_HEARTBEAT_INTERVAL`
command line example	`sensu-agent start --backend-heartbeat-interval 45`
/etc/sensu/agent.yml example	`backend-heartbeat-interval: 45`

backend-heartbeat-timeout

description	Number of seconds the agent should wait for a response to a hearbeat from the Sensu backend.
type	Integer
default	`45`
environment variable	`SENSU_BACKEND_HEARTBEAT_TIMEOUT`
command line example	`sensu-agent start --backend-heartbeat-timeout 60`
/etc/sensu/agent.yml example	`backend-heartbeat-timeout: 60`

backend-url

description	ws or wss URL of the Sensu backend server. To specify multiple backends with `sensu-agent start`, use this flag multiple times. *NOTE: If you do not specify a port for your backend-url values, the agent will automatically append the default backend port (8081).*
type	List
default	`ws://127.0.0.1:8081` (CentOS/RHEL, Debian, and Ubuntu) `$SENSU_HOSTNAME:8080` (Docker)
environment variable	`SENSU_BACKEND_URL`

command line example	**SHELL**
	
	```shell\nsensu-agent start --backend-url ws://127.0.0.1:8081\nsensu-agent start --backend-url ws://127.0.0.1:8081 --backend-url ws://127.0.0.1:8082\n```
	
	SHELL
	
	```shell\nsensu-agent start --backend-url wss://127.0.0.1:8081\nsensu-agent start --backend-url wss://127.0.0.1:8081 --backend-url wss://127.0.0.1:8082\n```

/etc/sensu/agent.yml example	**SHELL**
	
	```shell\nbackend-url:\n - "ws://127.0.0.1:8081"\n - "ws://127.0.0.1:8082"\n```
	
	SHELL
	
	```shell\nbackend-url:\n - "wss://127.0.0.1:8081"\n - "wss://127.0.0.1:8082"\n```

cache-dir

description	Path to store cached data.
type	String
default	- Linux: `/var/cache/sensu/sensu-agent` - Windows: `C:\ProgramData\sensu\cache\sensu-agent`
environment variable	`SENSU_CACHE_DIR`
command line example	`sensu-agent start --cache-dir /cache/sensu-agent`
/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	The config file path cannot be set by an environment variable.
command line	

example	
| | ```
sensu-agent start --config-file /sensu/agent.yml
sensu-agent start -c /sensu/agent.yml
``` |
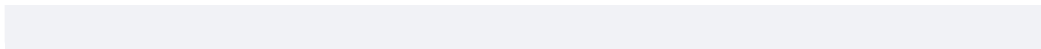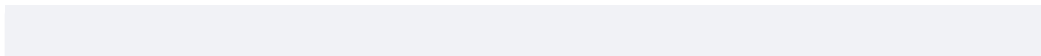
disable-assets

description	When set to `true`, disables dynamic runtime assets for the agent. If an agent attempts to execute a check that requires a dynamic runtime asset, the agent will respond with a status of `3` and a message that indicates the agent could not execute the check because assets are disabled.
type	Boolean
default	false
environment variable	`SENSU_DISABLE_ASSETS`
| command line example | ```
sensu-agent start --disable-assets
``` |
| /etc/sensu/agent.yml example | ```
disable-assets: true
``` |

discover-processes

description	When set to `true`, the agent populates the `processes` field in `entity.system` and updates every 20 seconds. **COMMERCIAL FEATURE**: Access the `discover-processes` flag in the packaged Sensu Go distribution. For more information, read Get started with commercial features. *NOTE: Process discovery is disabled in this version of Sensu. The `--discover-processes` flag is not available, and new events will not include data in the `processes` attributes. Instead, the field will be empty: `"processes": null`.*

type	Boolean
default	false
environment variable	`SENSU_DISCOVER_PROCESSES`
command line example	`sensu-agent start --discover-processes`
/etc/sensu/agent.yml example	`discover-processes: true`

labels

description	Custom attributes to include with event data that you can use for response and web UI view filtering.
	If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.
	Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
	NOTE: For labels that you define in agent.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the label `proxyType: "website"` *in agent.yml, it will be listed as* `proxytype: "website"` *in entity definitions.* *Key cases are **not** modified for labels you define with the* `--labels` *command line flag or the* `SENSU_LABELS` *environment variable.*
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8

	string.
default	`null`
environment variable	`SENSU_LABELS`
| command line example | ```
sensu-agent start --labels proxy_type=website
sensu-agent start --labels example_key1="example value"
example_key2="example value"
``` |
| /etc/sensu/agent.yml example | ```
labels:
  proxy_type: website
``` |

name

description	Entity name assigned to the agent entity.
type	String
default	Defaults to hostname (for example, `sensu-centos`).
environment variable	`SENSU_NAME`
| command line example | ```
sensu-agent start --name agent-01
``` |
| /etc/sensu/agent.yml example | ```
name: "agent-01"
``` |

log-level

description	Logging level: `panic` , `fatal` , `error` , `warn` , `info` , or `debug` .
type	String

default	`info`
environment variable	`SENSU_LOG_LEVEL`
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`
command line example	```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`
| command line example | ```
sensu-agent start --api-host 127.0.0.1
``` |
| /etc/sensu/agent.yml example | ```
api-host: "127.0.0.1"
``` |

api-port

description	Listening port for the Sensu agent HTTP API.
type	Integer
default	`3031`
environment variable	`SENSU_API_PORT`
| command line example | ```
sensu-agent start --api-port 3031
``` |
| /etc/sensu/agent.yml example | ```
api-port: 3031
``` |

disable-api

description	`true` to disable the agent HTTP API. Otherwise, `false`.
type	Boolean

default	`false`
environment variable	`SENSU_DISABLE_API`
command line example	`sensu-agent start --disable-api`
/etc/sensu/agent.yml example	`disable-api: true`

events-burst-limit

description	Maximum amount of burst allowed in a rate interval for the <u>agent events API</u>.
type	Integer
default	`10`
environment variable	`SENSU_EVENTS_BURST_LIMIT`
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 <u>agent events API</u>.
type	Float
default	`10.0`

environment variable	`SENSU_EVENTS_RATE_LIMIT`
command line example	`sensu-agent start --events-rate-limit 20.0`
/etc/sensu/agent.yml example	`events-rate-limit: 20.0`

Ephemeral agent configuration flags

deregister	
description	`true` if a deregistration event should be created upon Sensu agent process stop. Otherwise, `false`. **NOTE**: *To receive alerts for failing keepalives, set to* `false`.
type	Boolean
default	`false`
environment variable	`SENSU_DEREGISTER`
command line example	`sensu-agent start --deregister`
/etc/sensu/agent.yml example	`deregister: true`

deregistration-handler	
description	Name of the event handler to use when processing the agent's deregistration events. This flag overrides any handlers applied by the

`deregistration-handler` backend configuration flag.

type	String
environment variable	`SENSU_DEREGISTRATION_HANDLER`
command line example	```sensu-agent start --deregistration-handler deregister```
/etc/sensu/agent.yml example	```deregistration-handler: deregister```

detect-cloud-provider

description	`true` to enable cloud provider detection mechanisms. Otherwise, `false`. When this flag is enabled, the agent will attempt to read files, resolve hostnames, and make HTTP requests to determine what cloud environment it is running in.
type	Boolean
default	`false`
environment variable	`SENSU_DETECT_CLOUD_PROVIDER`
command line example	```sensu-agent start --detect-cloud-provider false```
/etc/sensu/agent.yml example	```detect-cloud-provider: false```

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` .

> **NOTE**: The agent maps the `keepalive-critical-timeout` value to the `event.check.ttl` attribute when keepalive events are generated for the Sensu backend to process. The `event.check.ttl` attribute is useful for creating time-based event filters to reduce alert fatigue for agent keepalive events.

type	Integer
default	`0`
environment variable	`SENSU_KEEPALIVE_CRITICAL_TIMEOUT`
command line example	`sensu-agent start --keepalive-critical-timeout 300`
/etc/sensu/agent.yml example	`keepalive-critical-timeout: 300`

keepalive-handlers

description	Keepalive event handlers to use for the entity, specified in a comma-delimited list. You can specify any configured handler and invoke the `keepalive-handlers` flag multiple times. If keepalive handlers are not specified, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI.
type	List
default	`keepalive`
environment variable	`SENSU_KEEPALIVE_HANDLERS`
command line	

example	`sensu-agent start --keepalive-handlers slack,email`
| /etc/sensu/agent.yml example | ```
keepalive-handlers:
- slack
- email
``` |

keepalive-interval

description	Number of seconds between keepalive events.
type	Integer
default	`20`
environment variable	`SENSU_KEEPALIVE_INTERNAL`
command line 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`.
	NOTE: The agent maps the `keepalive-warning-timeout` value to the `event.check.timeout` attribute when keepalive events are generated for the Sensu backend to process. The `event.check.timeout` attribute is useful for *creating time-based event filters* to reduce alert fatigue for

agent keepalive events.

type	Integer
default	`120`
environment variable	`SENSU_KEEPALIVE_WARNING_TIMEOUT`
command line example	`sensu-agent start --keepalive-warning-timeout 300`
/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 single namespace.*
type	String
default	`default`
environment variable	`SENSU_NAMESPACE`
command line example	`sensu-agent start --namespace ops`
/etc/sensu/agent.yml example	`namespace: ops`

user

description	Sensu RBAC username used by the agent. Agents require get, list, create, update, and delete permissions for events across all namespaces.
type	String
default	`agent`
environment variable	`SENSU_USER`
command line example	`sensu-agent start --user agent-01`
/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`
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`
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 mTLS authentication. Sensu supports certificate bundles (or chains) as long as the agent (or leaf) certificate is the *first* certificate in the bundle.
type	String
default	`""`
environment variable	`SENSU_CERT_FILE`
command line	

example	
| | ```
sensu-agent start --cert-file /path/to/tls/agent.pem
``` |

/etc/sensu/agent.yml example	
| | ```
cert-file: "/path/to/tls/agent.pem"
``` |

trusted-ca-file

description	SSL/TLS certificate authority.
type	String
default	`""`
environment variable	`SENSU_TRUSTED_CA_FILE`
| command line example | ```
sensu-agent start --trusted-ca-file /path/to/tls/ca.pem
``` |
| /etc/sensu/agent.yml example | ```
trusted-ca-file: "/path/to/tls/ca.pem"
``` |

key-file

description	Path to the agent key file used in mTLS authentication.
type	String
default	`""`
environment variable	`SENSU_KEY_FILE`
| command line example | ```
sensu-agent start --key-file /path/to/tls/agent-key.pem
``` |
| /etc/sensu/agent.yml | |

example	key-file: "/path/to/tls/agent-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`
command line example	`sensu-agent start --insecure-skip-tls-verify`
/etc/sensu/agent.yml example	`insecure-skip-tls-verify: true`

require-fips

description	Require Federal Information Processing Standard (FIPS) support in OpenSSL. Logs an error at Sensu agent startup if `true` but OpenSSL is not running in FIPS mode.
	NOTE: *The* `--require-fips` *flag is only available within the Linux amd64 OpenSSL-linked binary.* Contact Sensu *to request the builds for OpenSSL with FIPS support.*
type	Boolean

default	false
environment variable	`SENSU_REQUIRE_FIPS`
command line example	`sensu-agent start --require-fips`
/etc/sensu/agent.yml example	`require-fips: true`

require-openssl

description	Use OpenSSL instead of Go's standard cryptography library. Logs an error at Sensu agent startup if `true` but Go's standard cryptography library is loaded. **NOTE**: *The* `--require-openssl` *flag is only available within the Linux amd64 OpenSSL-linked binary. [Contact Sensu](#) to request the builds for OpenSSL with FIPS support.*
type	Boolean
default	false
environment variable	`SENSU_REQUIRE_OPENSSL`
command line example	`sensu-agent start --require-openssl`
/etc/sensu/agent.yml example	`require-openssl: 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`
command line example	```sensu-agent start --socket-host 127.0.0.1```
/etc/sensu/agent.yml example	```socket-host: "127.0.0.1"```

socket-port

description	Port the Sensu agent socket listens on.
type	Integer
default	`3030`
environment variable	`SENSU_SOCKET_PORT`
command line example	```sensu-agent start --socket-port 3030```
/etc/sensu/agent.yml example	```socket-port: 3030```

disable-sockets

description	`true` to disable the agent TCP and UDP event sockets. Othewise,

	`false` .
type	Boolean
default	`false`
environment variable	`SENSU_DISABLE_SOCKETS`
command line example	`sensu-agent start --disable-sockets`
/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`
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`
| command line example | ```
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`
| 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`
command line example	`sensu-agent start --statsd-metrics-host 127.0.0.1`
/etc/sensu/agent.yml example	`statsd-metrics-host: "127.0.0.1"`

statsd-metrics-port	
description	Port used for the StatsD metrics server.
type	Integer
default	`8125`
environment variable	`SENSU_STATSD_METRICS_PORT`
command line example	`sensu-agent start --statsd-metrics-port 8125`
/etc/sensu/agent.yml example	`statsd-metrics-port: 8125`

Allow list configuration commands

The allow list includes check and hook commands the agent can execute. Use the allow-list flag to specify the path to the yaml or json file that contains your allow list.

Use these commands to build your allow list configuration file.

exec	
description	Command to allow the Sensu agent to run as a check or a hook.
required	true
type	String

example	YML
	
	``yml
	exec: "/usr/local/bin/check_memory.sh"
	``
	
	JSON
	
	``json
	{
	  "exec": "/usr/local/bin/check_memory.sh"
	}
	``

sha512	
description	Checksum of the check or hook executable.
required	false
type	String

example	YML
	
	``yml
	sha512: 4f926bf4328...
	``
	
	JSON
	
	``json
	{
	  "sha512": "4f926bf4328..."
	}
	``

args

description	Arguments for the `exec` command.
required	true
type	Array

example

YML

```yml
args:
- foo
```

JSON

```json
{
  "args": ["foo"]
}
```

enable_env

description	`true` to enable environment variables. Otherwise, `false`.
required	false
type	Boolean

example

YML

```yml
enable_env: true
```

JSON

```json
{
  "enable_env": true
}
```

Example allow list configuration file

YML

```yaml
- 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

Instead of using configuration flags, you can use environment variables to configure your Sensu agent. Each agent configuration flag has an associated environment variable. You can also create your own environment variables, as long as you name them correctly and save them in the correct place. Here's how.

1. Create the files from which the `sensu-agent` service configured by our supported packages will read environment variables: `/etc/default/sensu-agent` for Debian/Ubuntu systems or `/etc/sysconfig/sensu-agent` for RHEL/CentOS systems.

   SHELL

   ```
   $ sudo touch /etc/default/sensu-agent
   ```

   SHELL

   ```
   $ sudo touch /etc/sysconfig/sensu-agent
   ```

2. Make sure the environment variable is named correctly. All environment variables controlling Sensu configuration begin with `SENSU_`.

To rename a configuration flag you wish to specify as an environment variable, prepend `SENSU_` , convert dashes to underscores, and capitalize all letters. For example, the environment variable for the flag `api-host` is `SENSU_API_HOST` .

For a custom test variable, the environment variable name might be `SENSU_TEST_VAR` .

3. Add the environment variable to the environment file (`/etc/default/sensu-agent` for Debian/Ubuntu systems or `/etc/sysconfig/sensu-agent` for RHEL/CentOS systems).

   In this example, the `api-host` flag is configured as an environment variable and set to `"0.0.0.0"` :

   SHELL

   ```
   $ echo 'SENSU_API_HOST="0.0.0.0"' | sudo tee -a /etc/default/sensu-agent
   ```

   SHELL

   ```
   $ echo 'SENSU_API_HOST="0.0.0.0"' | sudo tee -a /etc/sysconfig/sensu-agent
   ```

4. Restart the sensu-agent service so these settings can take effect.

   SHELL

   ```
   $ sudo systemctl restart sensu-agent
   ```

   SHELL

   ```
   $ sudo systemctl restart sensu-agent
   ```

> **NOTE**: Sensu includes an environment variable for each agent configuration flag. They are listed in the _configuration flag description tables_.

Format for label and annotation environment variables

To use labels and annotations as environment variables in your check and plugin configurations, you must use a specific format when you create the `SENSU_LABELS` and `SENSU_ANNOTATIONS` environment variables.

For example, to create the labels `"region": "us-east-1"` and `"type": "website"` as an environment variable:

SHELL

```
$ echo 'SENSU_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo tee -a
/etc/default/sensu-agent
```

SHELL

```
$ echo 'SENSU_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo tee -a
/etc/sysconfig/sensu-agent
```

To create the annotations `"maintainer": "Team A"` and `"webhook-url": "https://hooks.slack.com/services/T0000/B00000/XXXXX"` as an environment variable:

SHELL

```
$ echo 'SENSU_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/default/sensu-agent
```

SHELL

```
$ echo 'SENSU_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/sysconfig/sensu-agent
```

Use environment variables with the Sensu agent

Any environment variables you create in `/etc/default/sensu-agent` (Debian/Ubuntu) or `/etc/sysconfig/sensu-agent` (RHEL/CentOS) will be available to check and hook commands executed by the Sensu agent. This includes your checks and plugins.

For example, if you create a `SENSU_TEST_VAR` variable in your sensu-agent file, it will be available to use in your check configurations as `$SENSU_TEST_VAR`.

Use environment variables to specify an HTTP proxy for agent use

> **WARNING**: To use HTTP proxy environment variables, upgrade to Sensu Go 6.1.4 or later. In earlier versions of Sensu Go 6.1, the agent will not respect HTTP proxy environment variables when `trusted-ca-file` is configured. Upgrade to Sensu Go 6.1.4 or later to avoid this issue.

If an HTTP proxy is required to access the internet in your compute environment, you may need to configure the Sensu agent to successfully download dynamic runtime assets or execute commands that depend on internet access.

For Sensu agents that require a proxy server, define `HTTP_PROXY` and `HTTPS_PROXY` environment variables in your sensu-agent file.

```
HTTP_PROXY="http://YOUR_PROXY_SERVER:PORT"
HTTPS_PROXY="http://YOUR_PROXY_SERVER:PORT"
```

You can use the same proxy server URL for `HTTP_PROXY` and `HTTPS_PROXY`. The proxy server URL you specify for `HTTPS_PROXY` does not need to use `https://`.

After you add the `HTTP_PROXY` and `HTTPS_PROXY` environment variables and restart sensu-agent, they will be available to check and hook commands executed by the Sensu agent. You can then use `HTTP_PROXY` and `HTTPS_PROXY` to add dynamic runtime assets, run checks, and complete other tasks that typically require an internet connection for your unconnected entities.

> **NOTE**: If you define the `HTTP_PROXY` and `HTTPS_PROXY` environment variables, the agent WebSocket connection will also use the proxy URL you specify.

Backend reference

Example Sensu backend configuration file (download)

The Sensu backend is a service that manages check requests and observability data.Every Sensu backend includes an integrated structure for scheduling checks using subscriptions, an event processing pipeline that applies event filters, mutators, and handlers, an embedded etcd datastore for storing configuration and state, and the Sensu API, Sensu web UI, and sensuctl command line tool.

The Sensu backend is available for Ubuntu/Debian and RHEL/CentOS distributions of Linux.For these operating systems, the Sensu backend uses the Bourne shell (sh) for the execution environment.

Read the installation guide to install the backend.

Backend transport

The Sensu backend listens for agent communications via WebSocket transport.By default, this transport operates on port 8081.The agent subscriptions are used to determine which check execution requests the backend publishes via the transport.Sensu agents locally execute checks as requested by the backend and publish check results back to the transport to be processed.

Sensu agents authenticate to the Sensu backend via transport by either built-in username and password or mutual transport layer security (mTLS) authentication.

To secure the WebSocket transport, first generate the certificates you will need to set up transport layer security (TLS).Then, secure Sensu by configuring either TLS or mTLS to make Sensu production-ready.

Read the Sensu architecture overview for a diagram that includes the WebSocket transport.

Create event pipelines

Sensu backend event pipelines process observation data and executes event filters, mutators, and handlers.These pipelines are powerful tools to automate your monitoring workflows.To learn more abou event filters, mutators, and handlers, see:

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

- ¬ [Reduce alert fatigue with event filters](#)

- ¬ [Event 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, the backend database must be initialized by providing a username and password for the user to be granted administrative privileges. Although initialization is required for every new installation, the implementation differs depending on your method of installation:

- ¬ If you are using Docker, you can use environment variables to override the default admin username (`admin`) and password (`P@ssw0rd!`) during [step 2 of the backend installation process](#).

- ¬ If you are using Ubuntu/Debian or RHEL/CentOS, you must specify admin credentials during [step 3 of the backend installation process](#). Sensu does not apply a default admin username or password for Ubuntu/Debian or RHEL/CentoOS installations.

This step bootstraps the first admin user account for your Sensu installation. This account will be granted the cluster admin role.

> **WARNING**: If you plan to <u>run a Sensu cluster</u>, make sure that each of your backend nodes is configured, running, and a member of the cluster before you initialize.

Docker initialization

For Docker installations, set administrator credentials with environment variables when you <u>configure and start</u> the backend as shown below, replacing `YOUR_USERNAME` and `YOUR_PASSWORD` with the username and password you want to use:

DOCKER

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

DOCKER

```
---
version: "3"
services:
  sensu-backend:
    ports:
    - 3000:3000
    - 8080:8080
    - 8081:8081
    volumes:
    - "sensu-backend-data:/var/lib/sensu/sensu-backend/etcd"
    command: "sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug"
    environment:
    - SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=YOUR_USERNAME
    - SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=YOUR_PASSWORD
    image: sensu/sensu:latest

volumes:
  sensu-backend-data:
```

```
        driver: local
```

If you did not use environment variables to override the default admin credentials in step 2 of the backend installation process, we recommend changing your default admin password as soon as you have installed sensuctl.

Ubuntu/Debian or RHEL/CentOS initialization

For Ubuntu/Debian or RHEL/CentOS, set administrator credentials with environment variables at initialization 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:

```
sensu-backend init --interactive
```

You will receive prompts for your username and password in interactive mode:

```
Admin Username: YOUR_USERNAME
Admin Password: YOUR_PASSWORD
```

> **NOTE**: If you are already using Sensu, you do not need to initialize. Your installation has already seeded the admin username and password you have set up. Running `sensu-backend init` on a previously initialized cluster has no effect — it will not change the admin credentials.

To view 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 <u>configuration flags</u>:

```
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

To view 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 view 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.

To view sensu-backend commands, run:

```
sensu-backend help
```

To list options for a specific command (in this case, sensu-backend start), run:

```
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 via flags

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. Review 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.

Certificate bundles or chains

The Sensu backend supports all types of certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle. This is because the Go standard library assumes that the first certificate listed in the PEM file is the server certificate — the certificate that the program will use to show its own identity.

If you send the server certificate alone instead of sending the whole bundle with the server certificate first, you will receive a `certificate not signed by trusted authority` error. You must present the whole chain to the remote so it can determine whether it trusts the server certificate through the chain.

Certificate revocation check

The Sensu backend checks certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP) endpoints for mutual transport layer security (mTLS), etcd client, and etcd peer connections whose remote sides present X.509 certificates that provide CRL and OCSP revocation information.

Configuration summary

To view configuration information for the sensu-backend start command, run:

```
sensu-backend start --help
```

The response will list command information and configuration flags for sensu-backend start:

```
start the sensu backend

Usage:
  sensu-backend start [flags]

General Flags:
      --agent-auth-cert-file string       TLS certificate in PEM format for agent
certificate authentication
      --agent-auth-crl-urls strings        URLs of CRLs for agent certificate
authentication
      --agent-auth-key-file string        TLS certificate key in PEM format for
agent certificate authentication
      --agent-auth-trusted-ca-file string   TLS CA certificate bundle in PEM format
for agent certificate authentication
      --agent-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)
      --annotations stringToString        entity annotations map (default [])
      --api-listen-address string         address to listen on for api traffic
(default "[::]:8080")
      --api-request-limit int             maximum API request body size, in bytes
(default 512000)
      --api-url string                    url of the api to connect to (default
"http://localhost:8080")
      --assets-burst-limit int            asset fetch burst limit (default 100)
      --assets-rate-limit float          maximum number of assets fetched per
second
      --cache-dir string                  path to store cached data (default
"/var/cache/sensu/sensu-backend")
      --cert-file string                  TLS certificate in PEM format
  -c, --config-file string                path to sensu-backend config file (default
"/etc/sensu/backend.yml")
      --dashboard-cert-file string        dashboard TLS certificate in PEM format
      --dashboard-host string             dashboard listener host (default "[::]")
      --dashboard-key-file string         dashboard TLS certificate key in PEM
format
      --dashboard-port int                dashboard listener port (default 3000)
      --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 JWTs
    --jwt-public-key-file string           path to the PEM-encoded public key to use
to verify JWT signatures
    --keepalived-buffer-size int           number of incoming keepalives that can
be buffered (default 100)
    --keepalived-workers int               number of workers spawned for processing
incoming keepalives (default 100)
    --key-file string                      TLS certificate key in PEM format
    --labels stringToString                entity labels map (default [])
    --log-level string                     logging level [panic, fatal, error,
warn, info, debug, trace] (default "warn")
    --metrics-refresh-interval string      Go duration value (e.g. 1h5m30s) that
governs how often metrics are refreshed. (default "1m")
    --pipelined-buffer-size int            number of events to handle that can be
buffered (default 100)
    --pipelined-workers int                number of workers spawned for handling
events through the event pipeline (default 100)
    --require-fips                         indicates whether fips support should be
required in openssl
    --require-openssl                      indicates whether openssl should be
required instead of go's built-in crypto
  -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 clients (default [http://localhost:2379])
    --etcd-cert-file string                   path to the client server TLS cert
file
    --etcd-cipher-suites strings              list of ciphers to use for etcd
TLS configuration
    --etcd-client-cert-auth                   enable client cert authentication
```

```
      --etcd-client-urls string                 client URLs to use when operating
as an etcd client
      --etcd-discovery string                   discovery URL used to bootstrap
the cluster
      --etcd-discovery-srv string               DNS SRV record used to bootstrap
the cluster
      --etcd-election-timeout uint              time in ms a follower node will
go without hearing a heartbeat before attempting to become leader itself (default
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
      --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 etcd client URLs to
listen on (default [http://127.0.0.1:2379])
      --etcd-listen-peer-urls strings           list of URLs to listen on for
peer traffic (default [http://127.0.0.1:2380])
      --etcd-max-request-bytes uint             maximum etcd request size in
bytes (use with caution) (default 1572864)
      --etcd-name string                        name for this etcd node (default
"default")
      --etcd-peer-cert-file string              path to the peer server TLS cert
file
      --etcd-peer-client-cert-auth              enable peer client cert
authentication
      --etcd-peer-key-file string               path to the peer server TLS key
file
      --etcd-peer-trusted-ca-file string        path to the peer server TLS
trusted CA file
      --etcd-quota-backend-bytes int            maximum etcd database size in
bytes (use with caution) (default 4294967296)
      --etcd-trusted-ca-file string             path to the client server TLS
trusted CA cert file
      --no-embed-etcd                           don't embed etcd, use external
```

```
etcd instead
```

General configuration flags

> **NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.

annotations	
description	Non-identifying metadata to include with entity data for backend dynamic runtime assets (for example, handler and mutator dynamic runtime assets). **NOTE**: For annotations that you define in backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the annotation `webhookURL: "https://my-webhook.com"` in backend.yml, it will be listed as `webhookurl: "https://my-webhook.com"` in entity definitions. Key cases are **not** modified for annotations you define with the `--annotations` command line flag or the `SENSU_BACKEND_ANNOTATIONS` environment variable.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_BACKEND_ANNOTATIONS`
command line example	```sensu-backend start --annotations sensu.io/plugins/slack/config/webhook-url=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX sensu-backend start --annotations example-key="example value" --annotations example-key2="example value"```

/etc/sensu/backend.yml example	
| | ```
annotations:
  sensu.io/plugins/slack/config/webhook-url:
"https://hooks.slack.com/services/T00000000/B00000000/XXXXX
XXXXXXXXXXXXXXXXXXX"
``` |

api-listen-address

description	Address the API daemon will listen for requests on.
type	String
default	`[::]:8080`
environment variable	`SENSU_BACKEND_API_LISTEN_ADDRESS`
| command line example | ```
sensu-backend start --api-listen-address [::]:8080
``` |
| /etc/sensu/backend.yml example | ```
api-listen-address: "[::]:8080"
``` |

api-request-limit

description	Maximum size for API request bodies. In bytes.
type	Integer
default	`512000`
environment variable	`SENSU_BACKEND_API_REQUEST_LIMIT`
| command line example | ```
sensu-backend start --api-request-limit 1024000
``` |

/etc/sensu/backend.yml example	`api-request-limit: 1024000`

api-url

description	URL used to connect to the API.
type	String
default	`http://localhost:8080` (CentOS/RHEL, Debian, and Ubuntu) `http://$SENSU_HOSTNAME:8080` (Docker)
environment variable	`SENSU_BACKEND_API_URL`
command line example	`sensu-backend start --api-url http://localhost:8080`
/etc/sensu/backend.yml example	`api-url: "http://localhost:8080"`

assets-burst-limit

description	Maximum amount of burst allowed in a rate interval when fetching dynamic runtime assets.
type	Integer
default	`100`
environment variable	`SENSU_BACKEND_ASSETS_BURST_LIMIT`
command line example	`sensu-backend start --assets-burst-limit 100`

/etc/sensu/backend.yml example	`assets-burst-limit: 100`

assets-rate-limit

description	Maximum number of dynamic runtime assets to fetch per second. The default value `1.39` is equivalent to approximately 5000 user-to-server requests per hour.
type	Float
default	`1.39`
environment variable	`SENSU_BACKEND_ASSETS_RATE_LIMIT`
command line example	`sensu-backend start --assets-rate-limit 1.39`
/etc/sensu/backend.yml example	`assets-rate-limit: 1.39`

cache-dir

description	Path to store cached data.
type	String
default	`/var/cache/sensu/sensu-backend`
environment variable	`SENSU_BACKEND_CACHE_DIR`
command line example	`sensu-backend start --cache-dir /var/cache/sensu-backend`
/etc/sensu/backend.yml example	`cache-dir: "/var/cache/sensu-backend"`

config-file

description	Path to Sensu backend config file.
type	String
default	`/etc/sensu/backend.yml`
environment variable	The config file path cannot be set by an environment variable.
command line example	`sensu-backend start --config-file /etc/sensu/backend.yml` `sensu-backend start -c /etc/sensu/backend.yml`

debug

description	If `true`, enable debugging and profiling features for use with the Go pprof package. Otherwise, `false`.
type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_DEBUG`
command line example	`sensu-backend start --debug`
/etc/sensu/backend.yml example	`debug: true`

deregistration-handler

description	Name of the default event handler to use when processing agent deregistration events.

type	String
default	`""`
environment variable	`SENSU_BACKEND_DEREGISTRATION_HANDLER`
command line example	```sensu-backend start --deregistration-handler deregister```
/etc/sensu/backend.yml example	```deregistration-handler: "deregister"```

labels

description	Custom attributes to include with entity data for backend dynamic runtime assets (for example, handler and mutator dynamic runtime assets).
	NOTE: For labels that you define in backend.yml, the keys are automatically modified to use all lower-case letters. For example, if you define the label `securityZone: "us-west-2a"` in backend.yml, it will be listed as `securityzone: "us-west-2a"` in entity definitions. *Key cases are **not** modified for labels you define with the `--labels` command line flag or the `SENSU_BACKEND_LABELS` environment variable.*
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`
environment variable	`SENSU_BACKEND_LABELS`

| command line example | ```
sensu-backend start --labels security_zone=us-west-2a
sensu-backend start --labels example_key1="example value"
example_key2="example value"
``` |
|---|---|
| /etc/sensu/backend.yml example | ```
labels:
  security_zone: "us-west-2a"
  example_key1: "example value"
  example_key2: "example value"
``` |

log-level

description	Logging level: `panic`, `fatal`, `error`, `warn`, `info`, `debug`, or `trace`.
type	String
default	`warn`
environment variable	`SENSU_BACKEND_LOG_LEVEL`
| command line example | ```
sensu-backend start --log-level debug
``` |
| /etc/sensu/backend.yml example | ```
log-level: "debug"
``` |

metrics-refresh-interval

description	Interval at which Sensu should refresh metrics. In hours, minutes, seconds, or a combination — for example, `5m`, `1m30s`, and `1h10m30s` are all valid values.
type	String

default	`1m`
environment variable	`SENSU_BACKEND_METRICS_REFRESH_INTERVAL`
command line example	`sensu-backend start --metrics-refresh-interval 10s`
/etc/sensu/backend.yml example	`metrics-refresh-interval: "10s"`

state-dir

description	Path to Sensu state storage: `/var/lib/sensu/sensu-backend` .
type	String
required	true
environment variable	`SENSU_BACKEND_STATE_DIR`
command line example	`sensu-backend start --state-dir /var/lib/sensu/sensu-backend` `sensu-backend start -d /var/lib/sensu/sensu-backend`
/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. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.

type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_CERT_FILE`
command line example	`sensu-backend start --agent-auth-cert-file /path/to/tls/backend-1.pem`
/etc/sensu/backend.yml example	`agent-auth-cert-file: /path/to/tls/backend-1.pem`

agent-auth-crl-urls

description	URLs of CRLs for agent certificate authentication. The Sensu backend uses this list to perform a revocation check for agent mTLS.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_CRL_URLS`
command line example	`sensu-backend start --agent-auth-crl-urls http://localhost/CARoot.crl`
/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`
command line example	```sensu-backend start --agent-auth-key-file /path/to/tls/backend-1-key.pem```
/etc/sensu/backend.yml example	```agent-auth-key-file: /path/to/tls/backend-1-key.pem```

agent-auth-trusted-ca-file

description	TLS CA certificate bundle in PEM format for agent certificate authentication.
type	String
default	`""`
environment variable	`SENSU_BACKEND_AGENT_AUTH_TRUSTED_CA_FILE`
command line example	```sensu-backend start --agent-auth-trusted-ca-file /path/to/tls/ca.pem```
/etc/sensu/backend.yml example	```agent-auth-trusted-ca-file: /path/to/tls/ca.pem```

agent-host

description	Agent listener host. Listens on all IPv4 and IPv6 addresses by default.
type	String

default	`[::]`
environment variable	`SENSU_BACKEND_AGENT_HOST`
command line example	`sensu-backend start --agent-host 127.0.0.1`
/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`
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 web UI and end user web browsers, as well as communication between sensuctl and the Sensu API. Sensu supports certificate bundles (or chains) as long as the server

(or leaf) certificate is the *first* certificate in the bundle.

type	String
default	`""`
environment variable	`SENSU_BACKEND_CERT_FILE`
command line example	`sensu-backend start --cert-file /path/to/tls/backend-1.pem`
/etc/sensu/backend.yml example	`cert-file: "/path/to/tls/backend-1.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`
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`
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

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 web UI and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`
environment variable	`SENSU_BACKEND_KEY_FILE`
command line example	`sensu-backend start --key-file /path/to/tls/backend-1-key.pem`
/etc/sensu/backend.yml example	`key-file: "/path/to/tls/backend-1-key.pem"`

require-fips

description	Require Federal Information Processing Standard (FIPS) support in OpenSSL. Logs an error at Sensu backend startup if `true` but OpenSSL is not running in FIPS mode.

> **NOTE**: The `--require-fips` flag is only available within the Linux amd64 OpenSSL-linked binary. Contact Sensu to request the builds

	for OpenSSL with FIPS support.
type	Boolean
default	false
environment variable	`SENSU_BACKEND_REQUIRE_FIPS`
command line example	`sensu-backend start --require-fips`
/etc/sensu/backend.yml example	`require-fips: true`

require-openssl

description	Use OpenSSL instead of Go's standard cryptography library. Logs an error at Sensu backend startup if `true` but Go's standard cryptography library is loaded.
	NOTE: The `--require-openssl` flag is only available within the Linux amd64 OpenSSL-linked binary. *Contact Sensu* to request the builds for OpenSSL with FIPS support.
type	Boolean
default	false
environment variable	`SENSU_BACKEND_REQUIRE_OPENSSL`
command line example	`sensu-backend start --require-openssl`
/etc/sensu/backend.yml example	`require-openssl: true`

trusted-ca-file	
description	Path to the primary backend CA file. Specifies a fallback SSL/TLS certificate authority in PEM format used for etcd client (mutual TLS) communication if the `etcd-trusted-ca-file` is not used. This CA file is used in communication between the Sensu web UI and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`
environment variable	`SENSU_BACKEND_TRUSTED_CA_FILE`
command line example	`sensu-backend start --trusted-ca-file /path/to/tls/ca.pem`
/etc/sensu/backend.yml example	`trusted-ca-file: "/path/to/tls/ca.pem"`

Web UI configuration flags

dashboard-cert-file	
description	Web UI TLS certificate in PEM format. This certificate secures communication with the Sensu web UI. If the `dashboard-cert-file` is not provided in the backend configuration, Sensu uses the certificate specified in the `cert-file` flag for the web UI. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.
type	String
default	`""`

environment variable	`SENSU_BACKEND_DASHBOARD_CERT_FILE`
| command line example | ```
sensu-backend start --dashboard-cert-file
/path/to/tls/separate-webui-cert.pem
``` |
| /etc/sensu/backend.yml example | ```
dashboard-cert-file: "/path/to/tls/separate-webui-cert.pem"
``` |

dashboard-host

description	Web UI listener host.
type	String
default	`[::]`
environment variable	`SENSU_BACKEND_DASHBOARD_HOST`
| command line example | ```
sensu-backend start --dashboard-host 127.0.0.1
``` |
| /etc/sensu/backend.yml example | ```
dashboard-host: "127.0.0.1"
``` |

dashboard-key-file

description	Web UI TLS certificate key in PEM format. This key secures communication with the Sensu web UI. If the `dashboard-key-file` is not provided in the backend configuration, Sensu uses the key specified in the `key-file` flag for the web UI.
type	String
default	`""`

environment variable	`SENSU_BACKEND_DASHBOARD_KEY_FILE`
command line example	```sensu-backend start --dashboard-key-file /path/to/tls/separate-webui-key.pem```
/etc/sensu/backend.yml example	```dashboard-key-file: "/path/to/tls/separate-webui-key.pem"```

dashboard-port

description	Web UI listener port.
type	Integer
default	`3000`
environment variable	`SENSU_BACKEND_DASHBOARD_PORT`
command line example	```sensu-backend start --dashboard-port 3000```
/etc/sensu/backend.yml example	```dashboard-port: 3000```

Datastore and cluster configuration flags

> **NOTE**: Docker-only Sensu binds to the hostnames of containers, represented here as `SENSU_HOSTNAME` in Docker default values.

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` (CentOS/RHEL, Debian, and Ubuntu)
	`http://$SENSU_HOSTNAME:2379` (Docker)
environment variable	`SENSU_BACKEND_ETCD_ADVERTISE_CLIENT_URLS`
command line example	`sensu-backend start --etcd-advertise-client-urls http://localhost:2378,http://localhost:2379` `sensu-backend start --etcd-advertise-client-urls http://localhost:2378 --etcd-advertise-client-urls http://localhost:2379`
/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. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.

type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_CERT_FILE`
command line example	```sensu-backend start --etcd-cert-file /path/to/tls/backend-1.pem```
/etc/sensu/backend.yml example	```etcd-cert-file: "/path/to/tls/backend-1.pem"```

etcd-cipher-suites

description	List of allowed cipher suites for etcd TLS configuration. Sensu supports TLS 1.0-1.2 cipher suites as listed in the Go TLS documentation. You can use this attribute to defend your TLS servers from attacks on weak TLS ciphers. Go determines the default cipher suites based on the hardware used. *NOTE: To use TLS 1.3, add the following environment variable: `GODEBUG="tls13=1"`.* *To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

recommended	```etcd-cipher-suites:
	  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
	  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
	  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
	  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
	  - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305```

```
    - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
```

type	List

environment variable	`SENSU_BACKEND_ETCD_CIPHER_SUITES`

command line example	`sensu-backend start --etcd-cipher-suites TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` `sensu-backend start --etcd-cipher-suites TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 --etcd-cipher-suites TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

/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`. **NOTE**: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	Boolean

default	`false`

environment variable	`SENSU_BACKEND_ETCD_CLIENT_CERT_AUTH`

command line example	`sensu-backend start --etcd-client-cert-auth`

/etc/sensu/backend.yml example	
	`etcd-client-cert-auth: true`

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. **NOTE**: *To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (* `/etc/sensu/backend.yml` *).*

type	List
default	`http://127.0.0.1:2379`
environment variable	`SENSU_BACKEND_ETCD_CLIENT_URLS`

command line example	`sensu-backend start --etcd-client-urls`
	`'https://10.0.0.1:2379 https://10.1.0.1:2379'`
	`sensu-backend start --etcd-client-urls`
	`https://10.0.0.1:2379 --etcd-client-urls`
	`https://10.1.0.1:2379`

/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. **NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).
type	String
default	""
environment variable	`SENSU_BACKEND_ETCD_DISCOVERY`

command line example	`sensu-backend start --etcd-discovery`
	`https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579c`
	`bf573de`

/etc/sensu/backend.yml example	`etcd-discovery:`
	`  -`
	`https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579c`
	`bf573de`

etcd-discovery-srv

description	Exposes etcd's embedded auto-discovery features. Attempts to use a DNS SRV record to get the cluster configuration. **NOTE**: To use Sensu with an external etcd cluster, follow etcd's clustering guide. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file

(`/etc/sensu/backend.yml`).

type	String
default	""
environment variable	`SENSU_BACKEND_ETCD_DISCOVERY_SRV`
command line example	`sensu-backend start --etcd-discovery-srv example.org`
/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.
	NOTE: To use Sensu with an *external etcd cluster*, follow etcd's *clustering guide*.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	List
default	`http://127.0.0.1:2380` (CentOS/RHEL, Debian, and Ubuntu) `http://$SENSU_HOSTNAME:2380` (Docker)
environment variable	`SENSU_BACKEND_ETCD_INITIAL_ADVERTISE_PEER_URLS`
command line example	`sensu-backend start --etcd-initial-advertise-` `peer-urls`

```
https://10.0.0.1:2380,https://10.1.0.1:2380
sensu-backend start --etcd-initial-advertise-
peer-urls https://10.0.0.1:2380 --etcd-initial-
advertise-peer-urls https://10.1.0.1:2380
```

/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.

> **NOTE**: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	String
default	`default=http://127.0.0.1:2380` (CentOS/RHEL, Debian, and Ubuntu) `default=http://$SENSU_HOSTNAME:2380` (Docker)
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER`
| command line example | ```
sensu-backend start --etcd-initial-cluster backend-
0=https://10.0.0.1:2380,backend-
1=https://10.1.0.1:2380,backend-2=https://10.2.0.1:2380
``` |
| /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`).

> **NOTE**: *To use Sensu with an <u>external etcd cluster</u>, follow etcd's <u>clustering guide</u>.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	String
default	`new`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER_STATE`
command line example	```sensu-backend start --etcd-initial-cluster-state existing```
/etc/sensu/backend.yml example	```etcd-initial-cluster-state: "existing"```

etcd-initial-cluster-token

description	Unique token for the etcd cluster. Provide the same `etcd-initial-cluster-token` value for each cluster member. The `etcd-initial-cluster-token` allows etcd to generate unique cluster IDs and member IDs even for clusters with otherwise identical configurations, which prevents cross-cluster-interaction and potential cluster corruption.

> **NOTE**: *To use Sensu with an <u>external etcd cluster</u>, follow etcd's <u>clustering guide</u>.Do not configure external etcd in Sensu*

via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml` *).*

type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_INITIAL_CLUSTER_TOKEN`
command line example	`sensu-backend start --etcd-initial-cluster-token` `unique_token_for_this_cluster`
/etc/sensu/backend.yml example	`etcd-initial-cluster-token:` `"unique_token_for_this_cluster"`

etcd-key-file

description	Path to the etcd client API TLS key file. Secures communication between the embedded etcd client API and any etcd clients. **NOTE***: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (* `/etc/sensu/backend.yml` *).*
type	String
environment variable	`SENSU_BACKEND_ETCD_KEY_FILE`
command line example	`sensu-backend start --etcd-key-file /path/to/tls/backend-1-` `key.pem`

/etc/sensu/backend.yml example	etcd-key-file: "/path/to/tls/backend-1-key.pem"

etcd-listen-client-urls

description	List of URLs to listen on for client traffic. Sensu's default embedded etcd configuration listens for unencrypted client communication on port 2379.
	NOTE: To use Sensu with an external etcd cluster, follow etcd's clustering guide. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).
type	List
default	`http://127.0.0.1:2379` (CentOS/RHEL, Debian, and Ubuntu) `http://[::]:2379` (Docker)
environment variable	`SENSU_BACKEND_ETCD_LISTEN_CLIENT_URLS`
command line example	``` sensu-backend start --etcd-listen-client-urls https://10.0.0.1:2379,https://10.1.0.1:2379 sensu-backend start --etcd-listen-client-urls https://10.0.0.1:2379 --etcd-listen-client-urls https://10.1.0.1:2379 ```

/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. Sensu's default embedded etcd configuration listens for unencrypted peer communication on port 2380.
	NOTE: *To use Sensu with an* <u>*external etcd cluster*</u>*, follow etcd's* <u>*clustering guide*</u>*.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file* *(* `/etc/sensu/backend.yml` *).*

type	List

default	`http://127.0.0.1:2380` (CentOS/RHEL, Debian, and Ubuntu) `http://[::]:2380` (Docker)

environment variable	`SENSU_BACKEND_ETCD_LISTEN_PEER_URLS`

command line example	``` sensu-backend start --etcd-listen-peer-urls https://10.0.0.1:2380,https://10.1.0.1:2380 sensu-backend start --etcd-listen-peer-urls https://10.0.0.1:2380 --etcd-listen-peer-urls https://10.1.0.1:2380 ```

/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.
	NOTE: *To use Sensu with an* <u>*external etcd cluster*</u>*, follow etcd's* <u>*clustering guide*</u>*.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file* *(* `/etc/sensu/backend.yml` *).*

type	String
default	`default`
environment variable	`SENSU_BACKEND_ETCD_NAME`
command line example	`sensu-backend start --etcd-name backend-0`
/etc/sensu/backend.yml example	`etcd-name: "backend-0"`

etcd-peer-cert-file

description	Path to the peer server TLS certificate file. Sensu supports certificate bundles (or chains) as long as the server (or leaf) certificate is the *first* certificate in the bundle.
	NOTE: To use Sensu with an external etcd cluster, follow etcd's clustering guide. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_CERT_FILE`
command line example	`sensu-backend start --etcd-peer-cert-file /path/to/tls/backend-1.pem`
/etc/sensu/backend.yml example	`etcd-peer-cert-file: "/path/to/tls/backend-1.pem"`

etcd-peer-client-cert-auth

description	Enable peer client certificate authentication.

> **NOTE**: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_ETCD_PEER_CLIENT_CERT_AUTH`
command line example	`sensu-backend start --etcd-peer-client-cert-auth`
/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.

> **NOTE**: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_KEY_FILE`

command line example	sensu-backend start --etcd-peer-key-file /path/to/tls/backend-1-key.pem
/etc/sensu/backend.yml example	etcd-peer-key-file: "/path/to/tls/backend-1-key.pem"

etcd-peer-trusted-ca-file

description	Path to the etcd peer API server TLS trusted CA file. Secures communication between etcd cluster members.
	NOTE: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).
type	String
environment variable	`SENSU_BACKEND_ETCD_PEER_TRUSTED_CA_FILE`
command line example	sensu-backend start --etcd-peer-trusted-ca-file ./ca.pem
/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.

> **NOTE**: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	String
default	`""`
environment variable	`SENSU_BACKEND_ETCD_TRUSTED_CA_FILE`
command line example	`sensu-backend start --etcd-trusted-ca-file ./ca.pem`
/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`. > **NOTE**: To use Sensu with an _external etcd cluster_, follow etcd's _clustering guide_. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

type	Boolean
default	`false`
environment variable	`SENSU_BACKEND_NO_EMBED_ETCD`
command line example	`sensu-backend start --no-embed-etcd`

/etc/sensu/backend.yml example	`no-embed-etcd: true`

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`
command line example	`sensu-backend start --eventd-buffer-size 100`
/etc/sensu/backend.yml example	`eventd-buffer-size: 100`

eventd-workers	
description	Number of workers spawned for processing incoming events that are stored in the eventd buffer.
	WARNING: *Modify with caution. Increasing this value may result in greater CPU usage.*

type	Integer
default	`100`
environment variable	`SENSU_BACKEND_EVENTD_WORKERS`
command line example	`sensu-backend start --eventd-workers 100`
/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`
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`
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`
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`
| command line example | ```
sensu-backend start --pipelined-workers 100
``` |
| /etc/sensu/backend.yml example | ```
pipelined-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). Set to at least 10 times the etcd-heartbeat-interval. Read the etcd time parameter documentation for details and other considerations.
	WARNING: *Make sure to set the same election timeout value for all etcd members in one cluster. Setting different values for etcd members may reduce cluster stability.*

type	Integer
default	`1000`
environment variable	`SENSU_BACKEND_ETCD_ELECTION_TIMEOUT`
command line example	```sensu-backend start --etcd-election-timeout 1000```
/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. Read the etcd time parameter documentation for details and other considerations.

> *WARNING: Make sure to set the same heartbeat interval value for all etcd members in one cluster. Setting different values for etcd members may reduce cluster stability.*

> *NOTE: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).*

type	Integer

default	`100`
environment variable	`SENSU_BACKEND_ETCD_HEARTBEAT_INTERVAL`
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.
	NOTE: To use Sensu with an external etcd cluster, follow etcd's clustering guide.Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (/etc/sensu/backend.yml).
type	Integer
default	`1572864`
environment variable	`SENSU_BACKEND_ETCD_MAX_REQUEST_BYTES`
command line example	`sensu-backend start --etcd-max-request-bytes 1572864`
/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.* **NOTE**: *To use Sensu with an* <u>external etcd cluster</u>*, follow etcd's* <u>clustering guide</u>*. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (* `/etc/sensu/backend.yml` *).*
type	Integer
default	`4294967296`
environment variable	`SENSU_BACKEND_ETCD_QUOTA_BACKEND_BYTES`
command line example	`sensu-backend start --etcd-quota-backend-bytes` `4294967296`
/etc/sensu/backend.yml example	`etcd-quota-backend-bytes: ` `4294967296`

Configuration via environment variables

Instead of using configuration flags, you can use environment variables to configure your Sensu backend. Each backend configuration flag has an associated environment variable. You can also create

your own environment variables, as long as you name them correctly and save them in the correct place.Here's how.

1. Create the files from which the `sensu-backend` service configured by our supported packages will read environment variables: `/etc/default/sensu-backend` for Debian/Ubuntu systems or `/etc/sysconfig/sensu-backend` for RHEL/CentOS systems.

   SHELL

   ```
   $ sudo touch /etc/default/sensu-backend
   ```

   SHELL

   ```
   $ sudo touch /etc/sysconfig/sensu-backend
   ```

2. Make sure the environment variable is named correctly.All environment variables controlling Sensu backend configuration begin with `SENSU_BACKEND_`.

   To rename a configuration flag you wish to specify as an environment variable, prepend `SENSU_BACKEND_`, convert dashes to underscores, and capitalize all letters.For example, the environment variable for the flag `api-listen-address` is `SENSU_BACKEND_API_LISTEN_ADDRESS`.

   For a custom test variable, the environment variable name might be `SENSU_BACKEND_TEST_VAR`.

3. Add the environment variable to the environment file (`/etc/default/sensu-backend` for Debian/Ubuntu systems or `/etc/sysconfig/sensu-backend` for RHEL/CentOS systems).

   For example, to create `api-listen-address` as an environment variable and set it to `192.168.100.20:8080`:

   SHELL

   ```
   $ echo 'SENSU_BACKEND_API_LISTEN_ADDRESS=192.168.100.20:8080' | sudo tee -a /etc/default/sensu-backend
   ```

   SHELL

   ```
   $ echo 'SENSU_BACKEND_API_LISTEN_ADDRESS=192.168.100.20:8080' | sudo tee -a
   ```

```
/etc/sysconfig/sensu-backend
```

4. Restart the sensu-backend service so these settings can take effect.

    SHELL

    ```
    $ sudo systemctl restart sensu-backend
    ```

    SHELL

    ```
    $ sudo systemctl restart sensu-backend
    ```

> **NOTE**: Sensu includes an environment variable for each backend configuration flag. They are listed in the _configuration flag description tables_.

Format for label and annotation environment variables

To use labels and annotations as environment variables in your handler configurations, you must use a specific format when you create the `SENSU_BACKEND_LABELS` and `SENSU_BACKEND_ANNOTATIONS` environment variables.

For example, to create the labels `"region": "us-east-1"` and `"type": "website"` as an environment variable:

SHELL

```
$ echo 'SENSU_BACKEND_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo
tee -a /etc/default/sensu-backend
```

SHELL

```
$ echo 'SENSU_BACKEND_LABELS='{"region": "us-east-1", "type": "website"}'' | sudo
tee -a /etc/sysconfig/sensu-backend
```

To create the annotations `"maintainer": "Team A"` and `"webhook-url":`

`"https://hooks.slack.com/services/T0000/B00000/XXXXX"` as an environment variable:

SHELL

```
$ echo 'SENSU_BACKEND_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/default/sensu-backend
```

SHELL

```
$ echo 'SENSU_BACKEND_ANNOTATIONS='{"maintainer": "Team A", "webhook-url":
"https://hooks.slack.com/services/T0000/B00000/XXXXX"}'' | sudo tee -a
/etc/sysconfig/sensu-backend
```

Use environment variables with the Sensu backend

Any environment variables you create in `/etc/default/sensu-backend` (Debian/Ubuntu) or `/etc/sysconfig/sensu-backend` (RHEL/CentOS) will be available to handlers executed by the Sensu backend.

For example, if you create a `SENSU_BACKEND_TEST_VAR` variable in your sensu-backend file, it will be available to use in your handler configurations as `$SENSU_BACKEND_TEST_VAR`.

Event logging

COMMERCIAL FEATURE: Access event logging in the packaged Sensu Go distribution. For more information, read *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.

NOTE: Event logs do not include log messages produced by sensu-backend service. To write Sensu service logs to flat files on disk, read *Log Sensu services with systemd*.

event-log-buffer-size

description	Buffer size of the event logger. Corresponds to the maximum number of events kept in memory in case the log file is temporarily unavailable or more events have been received than can be written to the log file.
type	Integer
default	100000
environment variable	`SENSU_BACKEND_EVENT_LOG_BUFFER_SIZE`
command line example	```sensu-backend start --event-log-buffer-size 100000```
/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`
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

To manually rotate event logs, first rename (move) the current log file. Then, send the *SIGHUP* signal to the sensu-backend process so it creates a new log file and starts logging to it. Most Linux distributions include `logrotate` to automatically rotate log files as a standard utility, configured to run once per day by default.

Because event log files can grow quickly for larger Sensu installations, we recommend using `logrotate` to automatically rotate log files more frequently. To use the example log rotation configurations listed below, you may need to <u>configure</u> `logrotate` <u>to run once per hour</u>.

Log rotation for systemd

In this example, the `postrotate` script will reload the backend after log rotate is complete.

```
/var/log/sensu/events.log
{
   rotate 3
   hourly
   missingok
   notifempty
   compress
   postrotate
     /bin/systemctl reload sensu-backend.service > /dev/null 2>/dev/null || true
   endscript
}
```

Without the `postrotate` script, the backend will not reload. This will cause sensu-backend (and sensu-agent, if translated for the Sensu agent) to no longer write to the log file, even if logrotate recreates the log file.

Log rotation for sysvinit

```
/var/log/sensu/events.log
{
   rotate 3
```

```
  hourly
  missingok
  notifempty
  compress
  postrotate
    kill -HUP `cat /var/run/sensu/sensu-backend.pid 2> /dev/null` 2> /dev/null ||
true
  endscript
}
```

Checks reference

Checks work with Sensu agents to produce observability events automatically.You can use checks to monitor server resources, services, and application health as well as collect and analyze metrics.Read Monitor server resources to get started.Use Bonsai, the Sensu asset hub, to discover, download, and share Sensu check dynamic runtime assets.

Check example (minimum recommended attributes)

This example shows a check resource definition that includes the minimum recommended attributes.

> **NOTE**: The attribute `interval` is not required if a valid `cron` schedule is defined.Read scheduling for more information.

YML

```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
  }
}
```

Check commands

Each Sensu check definition specifies a command and the schedule at which it should be executed. Check commands are executable commands that the Sensu agent executes.

A command may include command line arguments for controlling the behavior of the command executable. Many common checks are available as dynamic runtime assets from Bonsai and support command line arguments so different check definitions can use the same executable.

> **NOTE**: Sensu advises against requiring root privileges to execute check commands or scripts. The Sensu user is not permitted to kill timed-out processes invoked by the root user, which could result in zombie processes.

Check command execution

All check commands are executed by Sensu agents as the `sensu` user. Commands must be executable files that are discoverable on the Sensu agent system (for example, installed in a system `$PATH` directory).

Check result specification

Although Sensu agents attempt to execute any command defined for a check, successful check result processing requires adherence to a simple specification.

- ¬ Result data is output to STDOUT or STDERR.

  - ¬ For service checks, this output is typically a human-readable message.

  - ¬ For metric checks, this output contains the measurements gathered by the check.

- ¬ Exit status code indicates state.

  - ¬ `0` indicates OK.

  - ¬ `1` indicates WARNING.

  - ¬ `2` indicates CRITICAL.

  - ¬ Exit status codes other than `0`, `1`, and `2` indicate an UNKNOWN or custom status

> **PRO TIP**: *If you're familiar with the* **Nagios** *monitoring system, you may recognize this specification — it is the same one that Nagios plugins use. As a result, you can use Nagios plugins with Sensu without any modification.*

At every execution of a check command, regardless of success or failure, the Sensu agent publishes the check's result for eventual handling by the **event processor** (the Sensu backend).

Check scheduling

The Sensu backend schedules checks and publishes check execution requests to entities via a publish/subscribe model. Checks have a defined set of subscriptions: transport topics to which the Sensu backend publishes check requests. Sensu entities become subscribers to these topics (called subscriptions) via their individual `subscriptions` attribute.
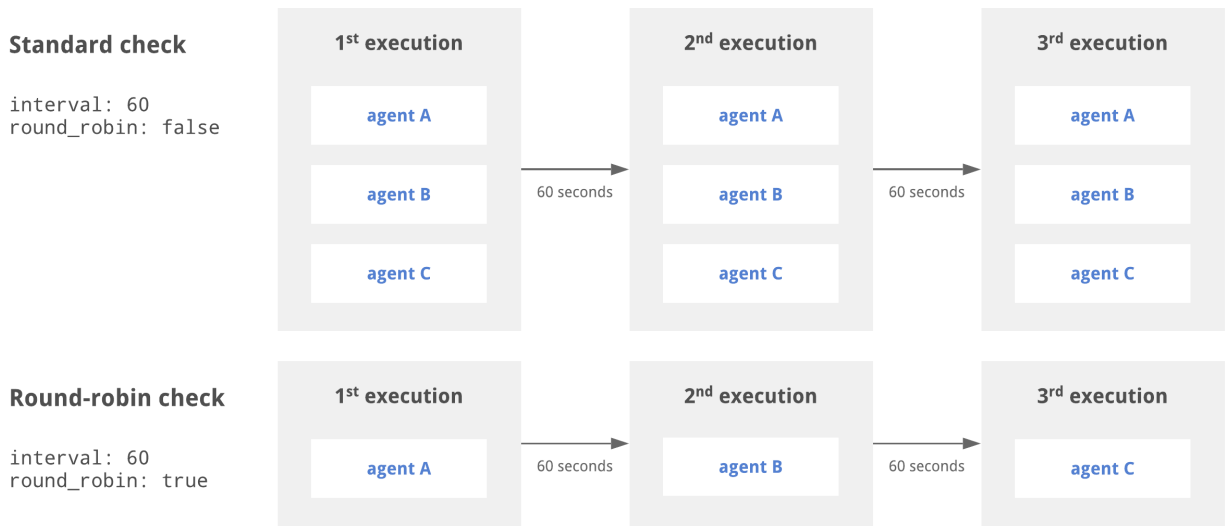
You can schedule checks using the `interval`, `cron`, and `publish` attributes. Sensu requires that checks include either an `interval` attribute (interval scheduling) or a `cron` attribute (cron scheduling).

Round robin checks

By default, Sensu schedules checks once per interval for each agent with a matching subscription: one

check execution per agent per interval.Sensu also supports deduplicated check execution when configured with the `round_robin` check attribute.For checks with `round_robin` set to `true`, Sensu executes the check once per interval, cycling through the available agents alphabetically according to agent name.

For example, for three agents configured with the `system` subscription (agents A, B, and C), a check configured with the `system` subscription and `round_robin` set to `true` results in one observability event per interval, with the agent creating the event following the pattern A -> B -> C -> A -> B -> C for the first six intervals.

Standard check

interval: 60
round_robin: false

1st execution		2nd execution		3rd execution
agent A		agent A		agent A
agent B	60 seconds	agent B	60 seconds	agent B
agent C		agent C		agent C

Round-robin check

interval: 60
round_robin: true

1st execution		2nd execution		3rd execution
agent A	60 seconds	agent B	60 seconds	agent C

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_.

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 (for example, `cron: '* * * * *'`).*

Example cron checks

To schedule a check to execute once a minute at the start of the minute, set the `cron` attribute to `* * * * *` and the `publish` attribute to `true`:

YML

```yaml
---
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

```yaml
---
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
  output_metric_tags: null
  proxy_entity_name: ""
```

```
    publish: true
    round_robin: false
    runtime_assets: null
    stdin: false
    subdue: null
    subscriptions:
    - sys
    timeout: 0
    ttl: 0
```

JSON

```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,
    "output_metric_tags": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "sys"
    ],
    "timeout": 0,
    "ttl": 0
```

```
      }
  }
```

Ad hoc scheduling

In addition to automatic execution, you can create checks to be scheduled manually using the <u>checks</u>
<u>API</u>.To create a check with ad-hoc scheduling, set the `publish` attribute to `false` in addition to an
`interval` or `cron` schedule.

Example ad hoc check
YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: ad_hoc_check
  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

```
---
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 observability events that represent each matching proxy entity. The entity attributes must match exactly as stated. No variables or directives have any special meaning, but you can 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. read the entity reference for information about using custom labels.

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  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 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 (for example, `0` , `1` , or `2`).

Learn how to use check hooks with the Sensu hooks reference documentation.

Check specification

> **NOTE**: In Sensu Go, the `occurrences` attribute is not part of the check definition like it was in Sensu Core.

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	**YML**
	
	``yaml
	type: CheckConfig
	``
	
	JSON
	
	``json
	{
	   "type": "CheckConfig"
	}
	``

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For

	checks in this version of Sensu, this attribute should always be `core/v2`.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

example

YML

```
api_version: core/v2
```

JSON

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

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

example

YML

```
metadata:
  name: collect-metrics
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
```

```
    slack-channel: "#monitoring"
```

JSON

```json
{
  "metadata": {
    "name": "collect-metrics",
    "namespace": "default",
    "created_by": "admin",
    "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

YML

example	

```yaml
spec:
  command: "/etc/sensu/plugins/check-chef-client.go"
  interval: 10
  publish: true
  subscriptions:
  - production
```

JSON

```json
{
```

```
    "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 | **YML**<br><br>```
  name: check-cpu
```<br><br>**JSON**<br><br>```
  {
    "name": "check-cpu"
  }
``` |

namespace	
description	Sensu RBAC namespace that the check belongs to.

required	false
type	String
default	`default`
example	**YML**

```yml
  namespace: production
```

JSON

```json
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the check or last updated the check. Sensu automatically populates the `created_by` field when the check is created or updated.
required	false
type	String
example	**YML**

```yml
  created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation data in events that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example

YML

```yml
labels:
  environment: development
  region: us-west-2
```

JSON

```json
{
  "labels": {
    "environment": "development",
    "region": "us-west-2"
  }
}
```

annotations

description	Non-identifying metadata to include with observation data in events that you can access with event filters. You can use annotations to add data

that's meaningful to people or external tools that interact with Sensu.

In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.

required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	**YML**

```yml
annotations:
  managed-by: ops
  playbook: www.example.url
```

JSON

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

Spec attributes

NOTE: *Spec attributes are not required when sending an HTTP* `POST` *request to the agent or backend /events API.When doing so, the spec attributes are listed as individual top-level attributes in the check definition instead.*

command	
description	Check command to be executed.
required	true

type	String

YML

example

```yml
command: /etc/sensu/plugins/check-chef-client.go
```

JSON

```json
{
  "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

YML

example

```yml
subscriptions:
- production
```

JSON

```json
{
  "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

YML

```yml
handlers:
- pagerduty
- email
```

JSON

```json
{
  "handlers": [
    "pagerduty",
    "email"
  ]
}
```

interval

description	How often the check is executed. In seconds.
required	true (unless `cron` is configured)
type	Integer

example

YML

```yml
interval: 60
```

JSON

```json
{
  "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 (for example, `cron: '* * * * *'`).*
required	true (unless `interval` is configured)
type	String

YML

example

```
cron: 0 0 * * *
```

JSON

```
{
  "cron": "0 0 * * *"
}
```

publish

description	`true` if check requests are published for the check. Otherwise, `false` .
required	false
type	Boolean
default	`false`

YML

example	
| | ```yaml
publish: false
``` |
| | **JSON** |
| | ```json
{
  "publish": false
}
``` |

timeout

description	Check execution duration timeout (hard stop). In seconds.
required	false
type	Integer
example	**YML**
| | ```yaml
timeout: 30
``` |
| | **JSON** |
| | ```json
{
  "timeout": 30
}
``` |

ttl

description	The time-to-live (TTL) until check results are considered stale. In seconds. If an agent stops publishing results for the check and the TTL expires, an event will be created for the agent's entity.
	The check `ttl` must be greater than the check `interval` and should allow enough time for the check execution and result processing to

complete. For example, for a check that has an `interval` of `60` (seconds) and a `timeout` of `30` (seconds), the appropriate `ttl` is at least `90` (seconds).

To use check `ttl` and `round_robin` together, your check configuration must also specify a `proxy_entity_name`. If you do not specify a `proxy_entity_name` when using check `ttl` and `round_robin` together, your check will stop executing.

> **NOTE**: Adding TTLs to checks adds overhead, so use the `ttl` attribute sparingly.

required	false
type	Integer

example	**YML**
	
	`ttl: 100`
	
	JSON
	
	`{`
	  `"ttl": 100`
	`}`

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`

YML

example
```yaml
stdin: true
```

JSON
```json
{
  "stdin": true
}
```

low_flap_threshold

description	Flap detection low threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios. Read the event reference to learn more about how Sensu uses the `low_flap_threshold` value.
required	false
type	Integer

YML

example
```yaml
low_flap_threshold: 20
```

JSON
```json
{
  "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. Read the event

reference to learn more about how Sensu uses the `high_flap_threshold` value.

required	true (if `low_flap_threshold` is configured)
type	Integer

example

YML

```yml
high_flap_threshold: 60
```

JSON

```json
{
  "high_flap_threshold": 60
}
```

runtime_assets

description	Array of Sensu dynamic runtime assets (names). Required at runtime for the execution of the `command`.
required	false
type	Array

example

YML

```yml
runtime_assets:
- ruby-2.5.0
```

JSON

```json
{
  "runtime_assets": [
    "ruby-2.5.0"
  ]
}
```

check_hooks

description	Array of check response types with respective arrays of <u>Sensu hook names</u>. Sensu hooks are commands run by the Sensu agent in response to the result of the check command execution. Hooks are executed in order of precedence based on their severity type: `1` to `255`, `ok`, `warning`, `critical`, `unknown`, and finally `non-zero`.
required	false
type	Array

example

YML

```yml
check_hooks:
- '0':
  - passing-hook
  - always-run-this-hook
- critical:
  - failing-hook
  - collect-diagnostics
  - always-run-this-hook
```

JSON

```json
{
  "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 (for example, a network switch).
required	false
type	String
validated	`\A[\w\.\-]+\z`

YML

example

```
proxy_entity_name: switch-dc-01
```

JSON

```
{
   "proxy_entity_name": "switch-dc-01"
}
```

proxy_requests

description	Assigns a check to run for multiple entities according to their `entity_attributes`. In the example below, the check executes for all entities with entity class `proxy` and the custom proxy type label `website`. Proxy requests are a great way to reuse check definitions for a group of entities. For more information, review the proxy requests specification and Monitor external resources.
required	false
type	Hash

YML

example	

```yaml
proxy_requests:
  entity_attributes:
  - entity.entity_class == 'proxy'
  - entity.labels.proxy_type == 'website'
  splay: true
  splay_coverage: 90
```

JSON

```json
{
  "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
	YML
example	

```yaml
silenced:
- "*:routers"
```

JSON

```json
{
  "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 | **YML**<br><br>```yml
env_vars:
- RUBY_VERSION=2.5.0
- CHECK_HOST=my.host.internal
```<br><br>**JSON**<br><br>```json
{
  "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)

`prometheus_text` (Prometheus Exposition Text)

When a check includes an `output_metric_format`, Sensu will extract the metrics from the check output and add them to the event data in Sensu metric format. Read Collect metrics with Sensu checks.

required	false
type	String
example	

YML

```
output_metric_format:
- graphite_plaintext
```

JSON

```
{
  "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. The Sensu InfluxDB handler provides an example.
required	false
type	Array
example	

YML

```
output_metric_handlers:
- influx-db
```

JSON

```json
{
  "output_metric_handlers": [
    "influx-db"
  ]
}
```

output_metric_tags

description	Custom tags to enrich metric points produced by check output metric extraction. One name/value pair make up a single tag. The `output_metric_tags` array can contain multiple tags. You can use check token substitution for the `value` attribute in output metric tags.
required	false
type	Array
| example | **YML**<br><br>```yaml
output_metric_tags:
- name: instance
  value: "{{ .name }}"
- name: prometheus_type
  value: gauge
- name: service
  value: "{{ .labels.service }}"
```<br><br>**JSON**<br><br>```json
{
  "output_metric_tags": [
    {
      "name": "instance",
      "value": "{{ .name }}"
    },
```|

```
      {
        "name": "prometheus_type",
        "value": "gauge"
      },
      {
        "name": "service",
        "value": "{{ .labels.service }}"
      }
    ]
  }
```

round_robin

description	When set to `true`, Sensu executes the check once per interval, cycling through each subscribing agent in turn. Read round robin checks for more information. Use the `round_robin` attribute with proxy checks to avoid duplicate events and distribute proxy check executions evenly across multiple agents. Read about proxy checks for more information. To use check `ttl` and `round_robin` together, your check configuration must also specify a `proxy_entity_name`. If you do not specify a `proxy_entity_name` when using check `ttl` and `round_robin` together, your check will stop executing.
required	false
type	Boolean
default	`false`

example	**YML**
	
	`round_robin: true`
	
	JSON
	
	`{`
	`  "round_robin": true`

```
    }
```

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.

YML

example	

```
subdue: null
```

JSON

```
{
  "subdue": null
}
```

secrets

description	Array of the name/secret pairs to use with command execution.
required	false
type	Array

YML

example	

```
secrets:
- name: ANSIBLE_HOST
  secret: sensu-ansible-host
- name: ANSIBLE_TOKEN
  secret: sensu-ansible-token
```

JSON

```
{
  "secrets": [
    {
      "name": "ANSIBLE_HOST",
      "secret": "sensu-ansible-host"
    },
    {
      "name": "ANSIBLE_TOKEN",
      "secret": "sensu-ansible-token"
    }
  ]
}
```

Proxy requests attributes

entity_attributes	
description	Sensu entity attributes to match entities in the registry using Sensu query expressions.
required	false
type	Array
example	**YML**

```yml
entity_attributes:
- entity.entity_class == 'proxy'
- entity.labels.proxy_type == 'website'
```

JSON

```json
{
  "entity_attributes": [
    "entity.entity_class == 'proxy'",
    "entity.labels.proxy_type == 'website'"
  ]
}
```

splay

description	`true` if proxy check requests should be splayed, published evenly over a window of time, determined by the check interval and a configurable splay coverage percentage. Otherwise, `false`. 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	**YML**

```
splay: true
```

JSON

```
{
  "splay": true
}
``` |

splay_coverage

description	**Percentage** of the check interval over which Sensu can execute the check for all applicable entities, as defined in the entity attributes. Sensu uses the splay coverage attribute to determine the 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

YML

```yaml
splay_coverage: 90
```

JSON

```json
{
  "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

YML

```yaml
max_output_size: 1024
```

JSON

```json
{
  "max_output_size": 1024
}
```

discard_output

description	If `true`, discard check output after extracting metrics. No check output will be sent to the Sensu backend. Otherwise, `false`.
required	false
type	Boolean
example	**YML**

```yml
discard_output: true
```

JSON

```json
{
  "discard_output": true
}
```

`output_metric_tags` *attributes*

name

description	Name for the output metric tag.
required	true
type	String
example	**YML**

```yml
name: instance
```

JSON

```json
{
  "name": "instance"
}
```

value

description	Value for the output metric tag. Use check token substitution syntax for the `value` attribute, with dot-notation access to any event attribute.
required	true
type	String
example	**YML**

```yml
value: {{ .name }}
```

JSON

```json
{
  "value": "{{ .name }}"
}
```

`secrets` *attributes*

name

description	Name of the secret defined in the executable command. Becomes the environment variable presented to the check. Read Use secrets management in Sensu for more information.
required	true
type	String
example	**YML**

```yml
name: ANSIBLE_HOST
```

JSON

```
{
  "name": "ANSIBLE_HOST"
}
```

secret

description	Name of the Sensu secret resource that defines how to retrieve the <u>secret</u>.
required	true
type	String

example	**YML**
	
	```secret: sensu-ansible-host```
	
	JSON
	
	```{ "secret": "sensu-ansible-host" }```

Metric check example

The following example shows the resource definition for a check that collects <u>metrics</u> in Nagios Performance Data format:

YML

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

```yaml
      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: nagios_perfdata
    output_metric_handlers:
    - prometheus_gateway
    output_metric_tags:
    - name: instance
      value: '{{ .name }}'
    - name: prometheus_type
      value: gauge
    - name: service
      value: '{{ .labels.service }}'
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets: null
    stdin: false
    subscriptions:
    - system
    timeout: 0
    ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "collect-metrics",
```

```json
      "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": "nagios_perfdata",
      "output_metric_handlers": [
        "prometheus_gateway"
      ],
      "output_metric_tags": [
        {
          "name": "instance",
          "value": "{{ .name }}"
        },
        {
          "name": "prometheus_type",
          "value": "gauge"
        },
        {
          "name": "service",
          "value": "{{ .labels.service }}"
        }
```

```
    ],
    "env_vars": null,
    "discard_output": true
  }
}
```

Check example that uses secrets management

The check in the following example uses secrets management to keep a GitHub token private. Learn more about secrets management for your Sensu configuration in the secrets and secrets providers references.

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: ping-github-api
  namespace: default
spec:
  check_hooks: null
  command: ping-github-api.sh $GITHUB_TOKEN
  secrets:
  - name: GITHUB_TOKEN
    secret: github-token-vault
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "ping-github-api",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "ping-github-api.sh $GITHUB_TOKEN",
```

```json
      "secrets": [
        {
          "name": "GITHUB_TOKEN",
          "secret": "github-token-vault"
        }
      ]
    }
  }
```

Check example with a PowerShell script command

If you use a PowerShell script in your check command, make sure to include the `-f` flag in the command. The `-f` flag ensures that the proper exit code is passed into Sensu. For example:

YML

```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

```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
    }
  }
```

The dynamic runtime asset reference includes an example check definition that uses the asset path to correctly capture exit status codes from PowerShell plugins distributed as dynamic runtime assets.

Metrics reference

Sensu Go offers built-in support for collecting and processing service and time-series metrics for your entire infrastructure.

In Sensu, metrics are an optional component of observation data in events.Sensu events may contain check execution results, metrics, or both.Certain inputs like the Sensu StatsD listener or patterns like the Prometheus collector pattern will create metrics-only events.Events can also include metrics from check output metric extraction.

Use Sensu handlers to process extracted metrics and route them to databases like Elasticsearch, InfluxDB, Grafana, and Graphite.You can also use Sensu's time-series and long-term event storage integrations to process service and time-series metrics.

> **NOTE**: This reference describes the metrics component of observation data included in Sensu events, which is distinct from the Sensu metrics API.For information about HTTP GET access to internal Sensu metrics, read our metrics API documentation.

Metric check example

This check definition collects metrics in Graphite Plaintext Protocol format and sends the collected metrics to a metrics handler configured with the Sensu Go Graphite Handler dynamic runtime asset:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: collect-metrics
  namespace: default
spec:
  check_hooks: null
  command: metrics-disk-usage.rb
  env_vars: null
```

```yaml
  handlers:
  - debug
  high_flap_threshold: 0
  interval: 30
  low_flap_threshold: 0
  output_metric_format: graphite_plaintext
  output_metric_handlers:
  - sensu-go-graphite-handler
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu-plugins/sensu-plugins-disk-checks
  - sensu/sensu-ruby-runtime
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - linux
  timeout: 0
  ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "collect-metrics",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "metrics-disk-usage.rb",
    "env_vars": null,
    "handlers": "debug",
    "high_flap_threshold": 0,
    "interval": 30,
    "low_flap_threshold": 0,
    "output_metric_format": "graphite_plaintext",
    "output_metric_handlers": [
```

```
      "sensu-go-graphite-handler"
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-disk-checks",
      "sensu/sensu-ruby-runtime"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "linux"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

Metric event example

The example metric check will produce events similar to this metric event:

YML

```
---
entity:
  entity_class: agent
  system:
    hostname: sensu-centos
    os: linux
    platform: centos
    platform_family: rhel
    platform_version: 7.5.1804
    network:
      interfaces:
      - name: lo
        addresses:
        - 127.0.0.1/8
```

```yaml
        - "::1/128"
      - name: eth0
        mac: '08:00:27:8b:c9:3f'
        addresses:
        - 10.0.2.15/24
        - fe80::2a24:13f4:6b:c0b8/64
      - name: eth1
        mac: '08:00:27:3d:ce:39'
        addresses:
        - 172.28.128.63/24
        - fe80::a00:27ff:fe3d:ce39/64
    arch: amd64
    libc_type: glibc
    vm_system: vbox
    vm_role: guest
    cloud_provider: ''
    processes:
  subscriptions:
  - entity:sensu-centos
  - linux
  last_seen: 1625000586
  deregister: false
  deregistration: {}
  user: agent
  redact:
  - password
  - passwd
  - pass
  - api_key
  - api_token
  - access_key
  - secret_key
  - private_key
  - secret
  metadata:
    name: sensu-centos
    namespace: default
  sensu_agent_version: 6.4.0
check:
  command: metrics-disk-usage.rb
  handlers:
  - debug
```

```yaml
high_flap_threshold: 0
interval: 60
low_flap_threshold: 0
publish: true
runtime_assets:
- sensu-plugins/sensu-plugins-disk-checks
- sensu/sensu-ruby-runtime
subscriptions:
- linux
proxy_entity_name: ''
check_hooks:
stdin: false
subdue:
ttl: 0
timeout: 0
round_robin: false
duration: 0.060007931
executed: 1625000586
history:
- status: 0
  executed: 1625000346
- status: 0
  executed: 1625000406
- status: 0
  executed: 1625000466
- status: 0
  executed: 1625000526
- status: 0
  executed: 1625000586
issued: 1625000586
output: |
  sensu-centos.disk_usage.root.used 1515 1625000586
  sensu-centos.disk_usage.root.avail 40433 1625000586
  sensu-centos.disk_usage.root.used_percentage 4 1625000586
  sensu-centos.disk_usage.root.dev.used 0 1625000586
  sensu-centos.disk_usage.root.dev.avail 485 1625000586
  sensu-centos.disk_usage.root.dev.used_percentage 0 1625000586
  sensu-centos.disk_usage.root.run.used 51 1625000586
  sensu-centos.disk_usage.root.run.avail 446 1625000586
  sensu-centos.disk_usage.root.run.used_percentage 11 1625000586
  sensu-centos.disk_usage.root.boot.used 130 1625000586
  sensu-centos.disk_usage.root.boot.avail 885 1625000586
```

```
      sensu-centos.disk_usage.root.boot.used_percentage 13 1625000586
      sensu-centos.disk_usage.root.home.used 33 1625000586
      sensu-centos.disk_usage.root.home.avail 20446 1625000586
      sensu-centos.disk_usage.root.home.used_percentage 1 1625000586
      sensu-centos.disk_usage.root.vagrant.used 79699 1625000586
      sensu-centos.disk_usage.root.vagrant.avail 874206 1625000586
      sensu-centos.disk_usage.root.vagrant.used_percentage 9 1625000586
    state: passing
    status: 0
    total_state_change: 0
    last_ok: 1625000586
    occurrences: 5
    occurrences_watermark: 5
    output_metric_format: graphite_plaintext
    output_metric_handlers: sensu-go-graphite-handler
    env_vars:
    metadata:
      name: collect-metrics
      namespace: default
    secrets:
    is_silenced: false
    scheduler: memory
  metrics:
    handlers:
    points:
    - name: sensu-centos.disk_usage.root.used
      value: 1515
      timestamp: 1625000586
      tags:
    - name: sensu-centos.disk_usage.root.avail
      value: 40433
      timestamp: 1625000586
      tags:
    - name: sensu-centos.disk_usage.root.used_percentage
      value: 4
      timestamp: 1625000586
      tags:
    - name: sensu-centos.disk_usage.root.dev.used
      value: 0
      timestamp: 1625000586
      tags:
    - name: sensu-centos.disk_usage.root.dev.avail
```

```yaml
    value: 485
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.dev.used_percentage
    value: 0
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.run.used
    value: 51
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.run.avail
    value: 446
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.run.used_percentage
    value: 11
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.boot.used
    value: 130
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.boot.avail
    value: 885
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.boot.used_percentage
    value: 13
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.home.used
    value: 33
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.home.avail
    value: 20446
    timestamp: 1625000586
    tags:
- name: sensu-centos.disk_usage.root.home.used_percentage
    value: 1
    timestamp: 1625000586
```

```
      tags:
  - name: sensu-centos.disk_usage.root.vagrant.used
    value: 79699
    timestamp: 1625000586
    tags:
  - name: sensu-centos.disk_usage.root.vagrant.avail
    value: 874206
    timestamp: 1625000586
    tags:
  - name: sensu-centos.disk_usage.root.vagrant.used_percentage
    value: 9
    timestamp: 1625000586
    tags:
metadata:
  namespace: default
id: 7468a597-bc3c-4ea7-899c-51c4d2992689
sequence: 5
timestamp: 1625000586
```

JSON

```
{
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
```

```json
          "addresses": [
            "10.0.2.15/24",
            "fe80::2a24:13f4:6b:c0b8/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:3d:ce:39",
          "addresses": [
            "172.28.128.63/24",
            "fe80::a00:27ff:fe3d:ce39/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "entity:sensu-centos",
    "linux"
  ],
  "last_seen": 1625000586,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
```

```json
      "name": "sensu-centos",
      "namespace": "default"
    },
    "sensu_agent_version": "6.4.0"
  },
  "check": {
    "command": "metrics-disk-usage.rb",
    "handlers": [
      "debug"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-disk-checks",
      "sensu/sensu-ruby-runtime"
    ],
    "subscriptions": [
      "linux"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 0.060007931,
    "executed": 1625000586,
    "history": [
      {
        "status": 0,
        "executed": 1625000346
      },
      {
        "status": 0,
        "executed": 1625000406
      },
      {
        "status": 0,
        "executed": 1625000466
```

```json
      },
      {
        "status": 0,
        "executed": 1625000526
      },
      {
        "status": 0,
        "executed": 1625000586
      }
    ],
    "issued": 1625000586,
    "output": "sensu-centos.disk_usage.root.used 1515 1625000586\nsensu-
centos.disk_usage.root.avail 40433 1625000586\nsensu-
centos.disk_usage.root.used_percentage 4 1625000586\nsensu-
centos.disk_usage.root.dev.used 0 1625000586\nsensu-centos.disk_usage.root.dev.avail
485 1625000586\nsensu-centos.disk_usage.root.dev.used_percentage 0
1625000586\nsensu-centos.disk_usage.root.run.used 51 1625000586\nsensu-
centos.disk_usage.root.run.avail 446 1625000586\nsensu-
centos.disk_usage.root.run.used_percentage 11 1625000586\nsensu-
centos.disk_usage.root.boot.used 130 1625000586\nsensu-
centos.disk_usage.root.boot.avail 885 1625000586\nsensu-
centos.disk_usage.root.boot.used_percentage 13 1625000586\nsensu-
centos.disk_usage.root.home.used 33 1625000586\nsensu-
centos.disk_usage.root.home.avail 20446 1625000586\nsensu-
centos.disk_usage.root.home.used_percentage 1 1625000586\nsensu-
centos.disk_usage.root.vagrant.used 79699 1625000586\nsensu-
centos.disk_usage.root.vagrant.avail 874206 1625000586\nsensu-
centos.disk_usage.root.vagrant.used_percentage 9 1625000586\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1625000586,
    "occurrences": 5,
    "occurrences_watermark": 5,
    "output_metric_format": "graphite_plaintext",
    "output_metric_handlers": "sensu-go-graphite-handler",
    "env_vars": null,
    "metadata": {
      "name": "collect-metrics",
      "namespace": "default"
    },
    "secrets": null,
```

```json
      "is_silenced": false,
      "scheduler": "memory"
    },
    "metrics": {
      "handlers": null,
      "points": [
        {
          "name": "sensu-centos.disk_usage.root.used",
          "value": 1515,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.avail",
          "value": 40433,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.used_percentage",
          "value": 4,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.dev.used",
          "value": 0,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.dev.avail",
          "value": 485,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.dev.used_percentage",
          "value": 0,
          "timestamp": 1625000586,
          "tags": null
        },
```

```json
      {
        "name": "sensu-centos.disk_usage.root.run.used",
        "value": 51,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.run.avail",
        "value": 446,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.run.used_percentage",
        "value": 11,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.boot.used",
        "value": 130,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.boot.avail",
        "value": 885,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.boot.used_percentage",
        "value": 13,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.home.used",
        "value": 33,
        "timestamp": 1625000586,
        "tags": null
      },
```

```
      {
        "name": "sensu-centos.disk_usage.root.home.avail",
        "value": 20446,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.home.used_percentage",
        "value": 1,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.vagrant.used",
        "value": 79699,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.vagrant.avail",
        "value": 874206,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.vagrant.used_percentage",
        "value": 9,
        "timestamp": 1625000586,
        "tags": null
      }
    ]
  },
  "metadata": {
    "namespace": "default"
  },
  "id": "7468a597-bc3c-4ea7-899c-51c4d2992689",
  "sequence": 5,
  "timestamp": 1625000586
}
```

Extract metrics from check output

The Sensu agent can extract metrics data from check command output and populate an event's metrics attribute before sending the event to the Sensu backend for processing.

To extract metrics from check output:

- The check `command` execution must output metrics in one of Sensu's supported output metric formats.

- The check must include the `output_metric_format` attribute with a value that specifies one of Sensu's supported output metric formats.

When a check includes correctly configured `command` and `output_metric_format` attributes, Sensu will extract the specified metrics from the check output and add them to the event data in the metrics attribute.

You can also configure the check `output_metric_handlers` attribute to use a Sensu handler that is equipped to handle Sensu metrics. Read the checks reference or InfluxDB handler guide to learn more.

Supported output metric formats

Sensu supports the following formats for check output metric extraction.

Graphite	
output metric format	`graphite_plaintext`
documentation	Graphite Plaintext Protocol
example	`local.random.diceroll 4 123456789`

InfluxDB

output metric format	`influxdb_line`
documentation	InfluxDB Line Protocol
example	

```
weather,location=us-midwest temperature=82
1465839830100400200
```

Nagios

output metric format	`nagios_perfdata`
documentation	Nagios Performance Data
example	

```
PING ok - Packet loss = 0%, RTA = 0.80 ms |
percent_packet_loss=0, rta=0.80
```

OpenTSDB

output metric format	`opentsdb_line`
documentation	OpenTSDB Data Specification
example	

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

Prometheus

output metric format	`prometheus_text`
documentation	Prometheus Exposition Text
example	

```
http_requests_total{method="post",code="200"} 1027
1395066363000
```

Enrich metrics with tags

Output metric tags are custom tags you can apply to enrich the metric points produced by check output metric extraction.

Use output metric tags for the output metric formats that do not natively support tags: Graphite Plaintext Protocol and Nagios Performance Data.

Values for output metric tags are passed through to the metric points produced by check output metric extraction for formats that natively support tags (InfluxDB Line Protocol, OpenTSDB Data Specification, and Prometheus Exposition Text).

You can use check token substitution for the value attribute to include any event attribute in an output metric tag. For example, this tag will list the `event.time` attribute:

```
"output_metric_tags": [
  {
    "name": "instance",
    "value": "{{ .entity.system.hostname }}"
  }
]
```

Process extracted and tagged metrics

Specify the event handlers you want to process your Sensu metrics in the check `output_metric_handlers` attribute. With these event handlers, you can route metrics to one or more databases for storing and visualizing metrics, like Elasticsearch, InfluxDB, Grafana, and Graphite.

Many of our most popular metrics integrations for time-series and long-term event storage include curated, configurable quick-start templates to integrate Sensu with your existing workflows. You can also use Bonsai, the Sensu asset hub, to discover, download, and share dynamic runtime assets for processing metrics.

In check definitions, the `output_metric_handlers` list for metrics event handlers is distinct and separate from the `handlers` list for status event handlers. Having separate handlers attributes allows you to use different workflows for metrics and status without applying conditional filter logic. The events reference includes an example event with check and metric data.

You do not need to add a mutator to your check definition to process metrics with an event handler. The metrics attribute format automatically reduces metrics data complexity so event handlers can process metrics effectively.

Validate metrics

If the check output is formatted correctly according to its `output_metric_format`, the metrics will be extracted in Sensu metric format and passed to the observability pipeline. The Sensu agent will log errors if it cannot parse the check output.

Use the debug handler example to write metric events to a file for inspection. To confirm that the check extracted metrics, inspect the event passed to the handler in the debug-event.json file. The event will include a top-level metrics section populated with metrics points arrays if the Sensu agent correctly ingested the metrics.

Metrics specification

The check specification describes metrics attributes in checks.

The event specification describes metrics attributes in events.

Subscriptions reference

Sensu uses the publish/subscribe model of communication. The publish/subscribe model is powerful in ephemeral or elastic infrastructures, where the names and numbers of things change over time.

Because Sensu uses the publish/subscribe model, you can write checks even if you don't know the specific names of the entities that should run the checks. Likewise, your entities do not need to know the specific names of the checks they should execute. The Sensu backend coordinates check execution for you by comparing the subscriptions you specify in your checks and entities to determine which entities should receive execution requests for a given check.

The diagram below shows how Sensu coordinates check execution based on subscriptions. For example, the `check_cpu` check includes the `system` subscription. All three entities include the `system` subscription, so all three entities will execute the `check_cpu` check. However, only the `server01` and `database01` entities will execute `check_sshd_process` — the `webserver01` entity does not include the `linux` subscription required to execute `check_sshd_process` .
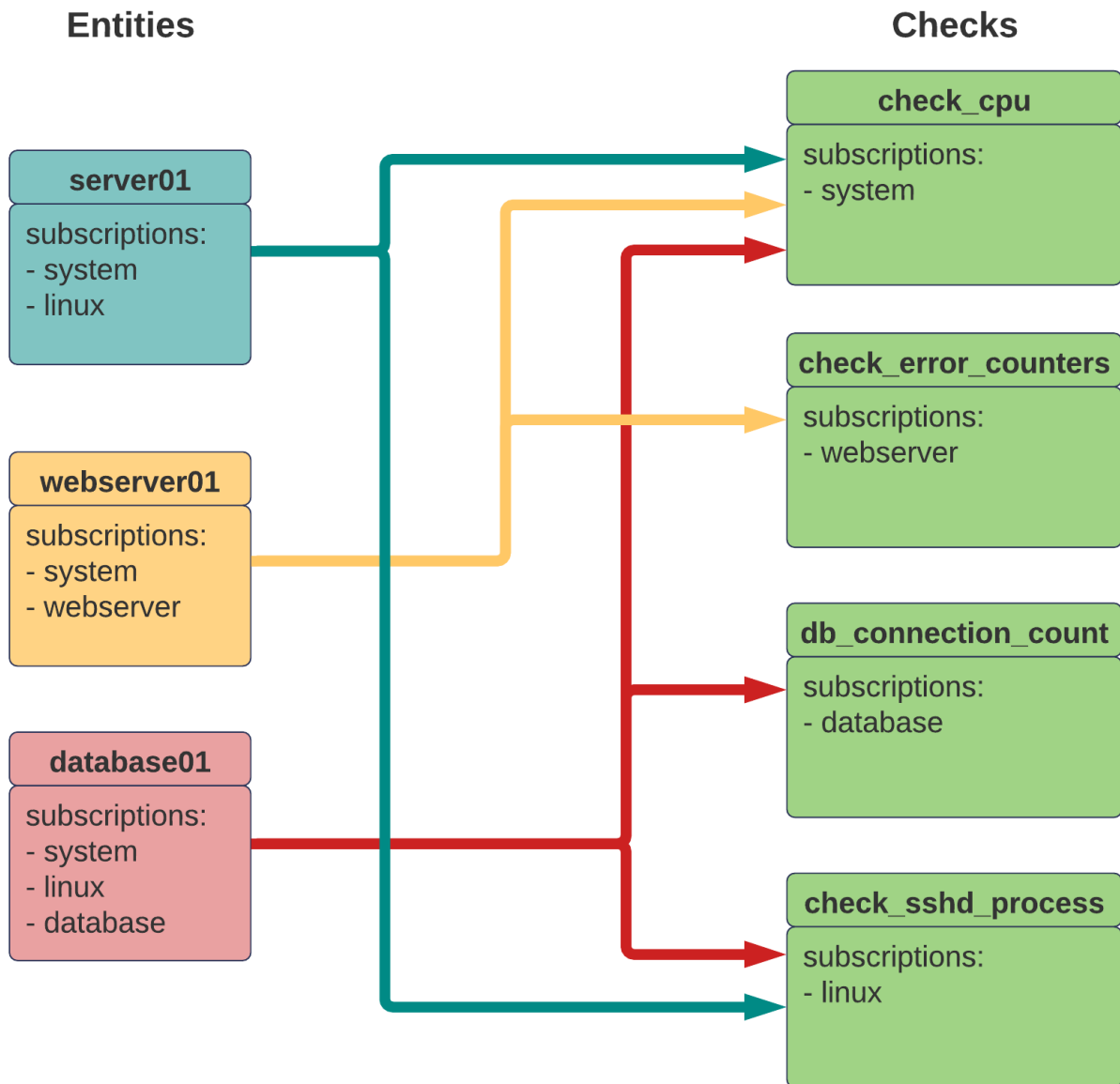
Entities

Checks

Sensu subscriptions are equivalent to topics in a traditional publish/subscribe system. Sensu entities become subscribers to these topics via the strings you specify with the agent `subscriptions` flag. Sensu checks have a `subscriptions` attribute, where you specify strings to indicate which subscribers will execute the checks. For Sensu to execute a check, the check definition must include a subscription that matches the subscription of at least one Sensu entity.

As loosely coupled references, subscriptions avoid the fragility of traditional host-based monitoring systems. Subscriptions allow you to configure check requests in a one-to-many model for entire groups or subgroups of entities rather than a traditional one-to-one mapping of configured hosts or observability checks.

Subscription example

Suppose you have a Sensu agent entity with the `linux` subscription:

```
sensu-agent start --subscriptions linux --log-level debug
```

For this agent to run a check, you must have at least one check with `linux` specified in the `subscriptions` attribute, such as this check to collect status information:

YML

```yaml
type: CheckConfig
api_version: core/v2
metadata:
  name: collect_info
  namespace: default
spec:
  command: collect.sh
  handlers:
  - slack
  interval: 10
  publish: true
  subscriptions:
  - linux
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "collect_info"
  },
  "spec": {
    "command": "collect.sh",
    "handlers": [
      "slack"
    ],
    "interval": 10,
```

```
      "publish": true,
      "subscriptions": [
        "linux"
      ]
    }
  }
```

If this is your only check for the `linux` subscription, this is the only check that your agent will execute. If you add more checks that specify the `linux` subscription, your agent will automatically run those checks too (as long as the `publish` attribute is set to `true` in the check definitions).

You can also add more subscriptions for your entity. For example, if you want your agent entity to execute checks for the `webserver` subscription, you can add it with the `subscriptions` flag:

```
sensu-agent start --subscriptions linux,webserver --log-level debug
```

Now your agent entity will execute checks with the `linux` or `webserver` subscriptions.

To directly add, update, and delete subscriptions for individual entities, use sensuctl, the entities API, or the web UI.

Configure subscriptions

Sensu automatically executes a check when the check definition includes a subscription that matches a subscription for at least one Sensu entity. In other words, subscriptions are configured for both checks and agent entities:

- ¬ To configure subscriptions for a check, add one or more subscription names in the check `subscriptions` attribute.
- ¬ To configure subscriptions for an agent entity, configure the `subscriptions` by specifying the subscriptions that include the checks the agent's entities should execute.

The Sensu backend schedules checks once per interval for each agent entity with a matching subscription. For example, if you have three entities configured with the `system` subscription, a check configured with the `system` subscription results in three monitoring events per interval: one check execution per entity per interval.

In addition to the subscriptions defined in the agent configuration, Sensu agent entities subscribe automatically to subscriptions that match their entity `name` . For example, an agent entity with `name: "i-424242"` subscribes to check requests with the subscription `entity:i-424242` . This makes it possible to generate ad hoc check requests that target specific entities via the API.

> **NOTE**: You can directly add, update, and delete subscriptions for individual entities via the backend with *sensuctl*, the *entities API*, and the *web UI*.

Publish checks

If you want Sensu to automatically schedule and execute a check according to its subscriptions, set the `publish` attribute to `true` in the check definition.

You can also manually schedule ad hoc check execution with the check API, whether the `publish` attribute is set to `true` or `false` . To target the subscriptions defined in the check, include only the check name in the request body (for example, `"check": "check_cpu"`). To override the check's subscriptions and target an alternate entity or group of entities, add the subscriptions attribute to the request body:

```
{
  "check": "check_cpu",
  "subscriptions": [
    "entity:i-424242",
    "entity:i-828282"
  ]
}
```

Monitor multiple servers

You can use subscriptions to configure monitoring and observability for multiple servers with different operating systems and monitoring requirements.

For example, suppose you want to set up monitoring for these servers:

- Six Linux servers:

¬ Get CPU, memory, and disk status for all six

¬ Get NGINX metrics for four

¬ Get PostgreSQL metrics for two

¬ Six Windows servers:

¬ Get CPU, memory, and disk checks for all six

¬ Get SQL Server metrics for two

This diagram shows the subscriptions to list for each of the 12 servers (the entities) and for each check to achieve the example monitoring configuration:

Entities

linux_1	linux_2	linux_3	linux_4	linux_5	linux_6
subscriptions: - linux	subscriptions: - linux	subscriptions: - linux - nginx	subscriptions: - linux - nginx	subscriptions: - linux - nginx - postgresql	subscriptions: - linux - nginx - postgresql

windows_1	windows_2	windows_3	windows_4	windows_5	windows_6
subscriptions: - windows	subscriptions: - windows	subscriptions: - windows	subscriptions: - windows	subscriptions: - windows - sqlserver	subscriptions: - windows - sqlserver

Checks

check_cpu	check_memory	check_disk	check_nginx	check_psql	check_sqlsrv
subscriptions: - linux - windows	subscriptions: - linux - windows	subscriptions: - linux - windows	subscriptions: - nginx	subscriptions: - postgresql	subscriptions: - sqlserver

In this scenario, none of the Windows servers should execute the NGINX metrics check, so the `check_nginx` subscriptions do not match any subscriptions listed for any of the Windows servers. Two of the six Windows servers *should* execute the SQL Server metrics check, so the subscription listed in the `check_sqlsrv` definition matches a subscription listed for those two Windows server entities.

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).This guide includes two check examples to help you monitor server resources (specifically, CPU usage and NGINX status).

To use this guide, you'll need to install a Sensu backend and have at least one Sensu agent running. Follow the RHEL/CentOS install instructions for the Sensu backend, the Sensu agent, and sensuctl.

Register dynamic runtime assets

You can write shell scripts in the `command` field of your check definitions, but we recommend using existing check plugins instead.Check plugins must be available on the host where the agent is running for the agent to execute the check.This guide uses dynamic runtime assets to manage plugin installation.

The Sensu CPU usage check dynamic runtime asset includes the `check-cpu-usage` command, which your CPU check will rely on.

To register the Sensu CPU usage check dynamic runtime asset, `sensu/check-cpu-usage:0.2.2`, run:

```
sensuctl asset add sensu/check-cpu-usage:0.2.2 -r check-cpu-usage
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/check-cpu-usage:0.2.2
added asset: sensu/check-cpu-usage:0.2.2
```

```
You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["check-cpu-usage"].
```

This example uses the `-r` (rename) flag to specify a shorter name for the dynamic runtime asset:
`check-cpu-usage`.

You can also download dynamic runtime asset definitions from <u>Bonsai</u> and register the asset with
`sensuctl create --file filename.yml`.

Then, use this command to register the <u>nagiosfoundation check plugin collection</u>, which you'll use later
for your webserver check:

```
sensuctl asset add ncr-devops-platform/nagiosfoundation:0.5.2 -r nagiosfoundation
```

To confirm that both dynamic runtime assets are ready to use, run:

```
sensuctl asset list
```

The response should list the `check-cpu-usage` and `nagiosfoundation` dynamic runtime assets:

```
   Name                      URL                              Hash
 _____

 _____ _____

 check-cpu-usage   //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_windows_amd64.tar.gz  900cfdf
 check-cpu-usage   //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_darwin_amd64.tar.gz   db81ee7
 check-cpu-usage   //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_armv7.tar.gz    400aacc
 check-cpu-usage   //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_arm64.tar.gz    bef7802
 check-cpu-usage   //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_386.tar.gz      a2dcb53
 check-cpu-usage   //assets.bonsai.sensu.io/.../check-cpu-usage_0.2.2_linux_amd64.tar.gz    2453973
 nagiosfoundation  //assets.bonsai.sensu.io/.../nagiosfoundation-windows-386-0.5.2.tgz      52edd59
 nagiosfoundation  //assets.bonsai.sensu.io/.../nagiosfoundation-windows-amd64-0.5.2.tgz    dfe121a
 nagiosfoundation  //assets.bonsai.sensu.io/.../nagiosfoundation-linux-386-0.5.2.tgz        a1791d7
 nagiosfoundation  //assets.bonsai.sensu.io/.../nagiosfoundation-linux-amd64-0.5.2.tgz      6b4f91b
```

Because plugins are published for multiple platforms, including Linux and Windows, the output will include multiple entries for each of the dynamic runtime assets.

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read *the asset reference* for more information about dynamic runtime asset builds.

Configure entity subscriptions

Every Sensu agent has a defined set of subscriptions that determine which checks the agent will execute. For an agent to execute a specific check, you must specify the same subscription in the agent configuration and the check definition. To run the CPU and NGINX webserver checks, you'll need a Sensu agent with the subscriptions `system` and `webserver`.

> **NOTE**: In production, your CPU and NGINX servers would be different entities, with the `system` subscription specified for the CPU entity and the `webserver` subscription specified for the NGINX entity. To keep things streamlined, this guide uses one entity to represent both.

To add the `system` and `webserver` subscriptions to the entity the Sensu agent is observing, first find your agent entity name:

```
sensuctl entity list
```

The `ID` is the name of your entity.

Replace `<entity_name>` with the name of your agent entity in the following sensuctl command. Run:

```
sensuctl entity update <entity_name>
```

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

Create a check to monitor a server

Now that the dynamic runtime assets are registered, create a check named `check_cpu` that runs the command `check-cpu-usage -w 75 -c 90` with the `check-cpu-usage` dynamic runtime asset at an interval of 60 seconds for all entities subscribed to the `system` subscription. This check generates a warning event (`-w`) when CPU usage reaches 75% and a critical alert (`-c`) at 90%.

```
sensuctl check create check_cpu \
--command 'check-cpu-usage -w 75 -c 90' \
--interval 60 \
--subscriptions system \
--runtime-assets check-cpu-usage
```

You should receive a confirmation message:

```
Created
```

To view the complete resource definition for `check_cpu` , run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The sensuctl response will include the complete `check_cpu` resource definition in the specified format:

YML

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

```
    created_by: admin
    name: check_cpu
    namespace: default
  spec:
    check_hooks: null
    command: check-cpu-usage -w 75 -c 90
    env_vars: null
    handlers: []
    high_flap_threshold: 0
    interval: 60
    low_flap_threshold: 0
    output_metric_format: ""
    output_metric_handlers: null
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
    - check-cpu-usage
    secrets: null
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_cpu",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
```

```
      "interval": 60,
      "low_flap_threshold": 0,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "secrets": null,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "ttl": 0
    }
  }
```

If you want to share, reuse, and maintain this check just like you would code, you can save it to a file and start building a monitoring as code repository.

> **PRO TIP**: You can also *view complete resource definitions in the Sensu web UI*.

Validate the CPU check

The Sensu agent uses WebSocket to communicate with the Sensu backend, sending event data as JSON messages.As your checks run, the Sensu agent captures check standard output (`STDOUT`) or standard error (`STDERR`).This data will be included in the JSON payload the agent sends to your Sensu backend as the event data.

It might take a few moments after you create the check for the check to be scheduled on the entity and the event to return to Sensu backend.Use sensuctl to view the event data and confirm that Sensu is monitoring CPU usage:

```
sensuctl event list
```

The response should list the `check_cpu` check, returning an OK status (`0`)

```
    Entity      Check                                                          Output
Status  Silenced        Timestamp                   UUID
_____ _____
_____
_____
_____ _____ _____ _____
_____
  sensu-centos   check_cpu   check-cpu-usage OK: 1.02% CPU usage | cpu_idle=98.98, cpu_system=0.51, cpu_user=0.51,
cpu_nice=0.00, cpu_iowait=0.00, cpu_irq=0.00, cpu_softirq=0.00, cpu_steal=0.00, cpu_guest=0.00, cpu_guestnice=0.00
0   false     2021-10-06 19:25:43 +0000 UTC   xxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Create a check to monitor a webserver

In this section, you'll create a check to monitor an NGINX webserver, similar to the CPU check you created in the previous section but using the `webserver` subscription rather than `system` .

Install and configure NGINX

The webserver check requires a running NGINX service, so you'll need to install and configure NGINX.

> **NOTE**: *You may need to install and update the EPEL repository with* `sudo yum install epel-release` *and* `sudo yum update` *before you can install NGINX.*

Install NGINX:

```
sudo yum install nginx
```

Enable and start the NGINX service:

```
systemctl enable nginx && systemctl start nginx
```

Verify that Nginx is serving webpages:

```
curl -sI http://localhost
```

The response should include `HTTP/1.1 200 OK` to indicates that NGINX processed your request as expected:

```
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Wed, 06 Oct 2021 19:35:14 GMT
Content-Type: text/html
Content-Length: 4833
Last-Modified: Fri, 16 May 2014 15:12:48 GMT
Connection: keep-alive
ETag: "xxxxxxxx-xxxx"
Accept-Ranges: bytes
```

With your NGINX service running, you can configure the webserver check.

Create the webserver check definition

Create a check that uses the `check_service` plugin from the nagiosfoundation collection. The `nginx_service` check will run at an interval of 15 seconds and determine whether the `nginx` service is running for all entities subscribed to the `webserver` subscription.

Run the following sensuctl command to create the `nginx_service` check:

```
sensuctl check create nginx_service \
--command 'check_service --name nginx' \
--interval 15 \
--subscriptions webserver \
--runtime-assets nagiosfoundation
```

You should receive a confirmation message:

```
Created
```

To view the complete resource definition for `nginx_service`, run:

SHELL

```
sensuctl check info nginx_service --format yaml
```

SHELL

```
sensuctl check info nginx_service --format wrapped-json
```

The sensuctl response will include the complete `nginx_service` resource definition in the specified format:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: nginx_service
  namespace: default
spec:
  check_hooks: null
  command: check_service --name nginx
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 15
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  proxy_entity_name: ""
  publish: true
```

```yaml
    round_robin: false
    runtime_assets:
    - nagiosfoundation
    secrets: null
    stdin: false
    subdue: null
    subscriptions:
    - webserver
    timeout: 0
    ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "nginx_service",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check_service --name nginx",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 15,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "nagiosfoundation"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "webserver"
```

```
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

As with the `check_cpu` check, you can share, reuse, and maintain this check just like code.

Validate the webserver check

It might take a few moments after you create the check for the check to be scheduled on the entity and the event to return to Sensu backend. Use sensuctl to view event data and confirm that Sensu is monitoring the NGINX webserver status:

```
sensuctl event list
```

The response should list the `nginx_service` check, returning an OK status (0):

Entity	Check	Output	Status	Silenced	Timestamp	UUID
sensu-centos	nginx_service	CheckService OK - nginx in a running state	0	false	2021-10-06 19:41:34 +0000 UTC	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Simulate a critical event

To manually generate a critical event for your `nginx_service` check, stop the NGINX service. Run:

```
systemctl stop nginx
```

When you stop the service, the check will generate a critical event. After a few moments, run:

```
sensuctl event list
```

The response should list the `nginx_service` check, returning a CRITICAL status (`2`):

```
   Entity      Check                        Output                      Status  Silenced        Timestamp
UUID
_____ _____
_____ _____
_____ _____ _____
 sensu-centos  nginx_service  CheckService CRITICAL - nginx not in a running state  (State: inactive)     2  false
2021-10-06 19:43:34 +0000 UTC  5118e6c5-4f9f-4f0e-8ad6-d42fd5033819
```

Restart the NGINX service to clear the event:

```
systemctl start nginx
```

After a moment, you can verify that the event cleared:

```
sensuctl event list
```

The response should list the `nginx_service` check with an OK status (`0`).

Next steps

Now that you know how to create checks to monitor CPU usage and NGINX webserver status, read the checks reference and assets reference for more detailed information.Or, learn how to monitor external resources with proxy checks and entities.

You can also create a handler to send alerts to email, PagerDuty, or Slack based on the status events your checks are generating.

Collect service 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 checks reference and Monitor server resources.

This guide demonstrates how to collect disk usage metrics with a check named `collect-metrics` and configure the check to extract metrics output in Graphite Plaintext Protocol format. To use this guide, you'll need to install a Sensu backend and have at least one Sensu agent running on Linux. In this guide, the Sensu agent is named `sensu-centos`.

Register dynamic runtime assets

To power the check to collect disk usage metrics, add the Sensu Disk Checks Plugin dynamic runtime asset. The Sensu Disk Checks Plugin asset includes the `metrics-disk-usage.rb` plugin, which your check will rely on.

The Sensu assets packaged from the Sensu Disk Checks Plugin asset are built against the Sensu Ruby runtime environment, so you also need to add the Sensu Go Ruby Runtime Assets dynamic runtime asset. The Sensu Ruby Runtime asset delivers the Ruby executable and supporting libraries the check will need to run the `metrics-disk-usage.rb` plugin.

Use sensuctl to register the Sensu Disk Checks Plugin dynamic runtime asset, `sensu-plugins/sensu-plugins-disk-checks`:

```
sensuctl asset add sensu-plugins/sensu-plugins-disk-checks:5.1.4
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu-plugins/sensu-plugins-disk-checks:5.1.4
added asset: sensu-plugins/sensu-plugins-disk-checks:5.1.4

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
```

```
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-plugins/sensu-plugins-
disk-checks"].
```

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

Then, use the following sensuctl example to register the Sensu Go Ruby Runtime Assets dynamic
runtime asset, `sensu/sensu-ruby-runtime` :

```
sensuctl asset add sensu/sensu-ruby-runtime:0.1.0
```

Again, you will receive an `added asset` response.

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

```
sensuctl asset list
```

The sensuctl response should list `sensu-plugins/disk-checks-plugins` and `sensu/sensu-ruby-
runtime` :

```
          Name                                     URL                           Hash
 _____
 _____
 _____ _____
  sensu-plugins/sensu-plugins-disk-checks  //assets.bonsai.sensu.io/.../sensu-plugins-disk-
 checks_5.1.4_centos8_linux_amd64.tar.gz    ac0c130
  sensu-plugins/sensu-plugins-disk-checks  //assets.bonsai.sensu.io/.../sensu-plugins-disk-
 checks_5.1.4_centos7_linux_amd64.tar.gz    81af33b
  sensu-plugins/sensu-plugins-disk-checks  //assets.bonsai.sensu.io/.../sensu-plugins-disk-
 checks_5.1.4_centos6_linux_amd64.tar.gz    e909a10
  sensu-plugins/sensu-plugins-disk-checks  //assets.bonsai.sensu.io/.../sensu-plugins-disk-
 checks_5.1.4_debian_linux_amd64.tar.gz     a10b824
  sensu-plugins/sensu-plugins-disk-checks  //assets.bonsai.sensu.io/.../sensu-plugins-disk-
```

```
checks_5.1.4_alpine_linux_amd64.tar.gz          cb95031
  sensu/sensu-ruby-runtime                  //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-
2.4.4_amzn2_linux_amd64.tar.gz    a83aaa5
  sensu/sensu-ruby-runtime                  //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-
2.4.4_amzn1_linux_amd64.tar.gz    7b504f0
  sensu/sensu-ruby-runtime                  //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-
2.4.4_centos8_linux_amd64.tar.gz  db4769f
  sensu/sensu-ruby-runtime                  //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-
2.4.4_centos7_linux_amd64.tar.gz  2d78004
  sensu/sensu-ruby-runtime                  //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-
2.4.4_debian_linux_amd64.tar.gz   956806a
  sensu/sensu-ruby-runtime                  //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-
2.4.4_alpine_linux_amd64.tar.gz   15af366
```

NOTE: *Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read* [the asset reference](#) *for more information about dynamic runtime asset builds.*

Create a check to collect metrics

The Sensu Plugins Disk Checks dynamic runtime asset includes the `metrics-disk-usage.rb` plugin. To use this plugin, create the `collect-metrics` check with a `metrics-disk-usage.rb` command:

```
sensuctl check create collect-metrics \
--command 'metrics-disk-usage.rb' \
--interval 60 \
--subscriptions linux \
--runtime-assets sensu-plugins/sensu-plugins-disk-checks,sensu/sensu-ruby-runtime \
--output-metric-format graphite_plaintext
```

This example check specifies a 60-second interval for collecting metrics, a subscription to ensure the check will run on any entity that includes the `linux` subscription, and the names of the two dynamic runtime assets the check needs to work properly.The check definition also specifies that the output metric format for the collected metrics should be `graphite_plaintext` (although you can use any of the supported output metric formats).

You should receive a confirmation response: `Created` .

To view the check resource you just created with sensuctl, run:

SHELL

```shell
sensuctl check info collect-metrics --format yaml
```

SHELL

```shell
sensuctl check info collect-metrics --format wrapped-json
```

The sensuctl response will list the complete check resource definition — you can add it to your monitoring as code repository:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: collect-metrics
  namespace: default
spec:
  check_hooks: null
  command: metrics-disk-usage.rb
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: graphite_plaintext
  output_metric_handlers: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu-plugins/sensu-plugins-disk-checks
  - sensu/sensu-ruby-runtime
  secrets: null
  stdin: false
```

```
  subdue: null
  subscriptions:
  - linux
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "collect-metrics",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "metrics-disk-usage.rb",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "graphite_plaintext",
    "output_metric_handlers": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-disk-checks",
      "sensu/sensu-ruby-runtime"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "linux"
    ],
    "timeout": 0,
    "ttl": 0
```

```
    }
  }
```

Confirm that your check is collecting metrics

If the check output is formatted correctly according to its `output_metric_format` , the metrics will be extracted in Sensu metric format and passed to the observability pipeline for handling. The Sensu agent will log errors if it cannot parse the check output.

Add a debug handler to write metric events to a file for inspection. To confirm that the check extracted metrics, inspect the event passed to the handler in the debug-event.json file. The event will include a top-level metrics section populated with metrics points arrays if the Sensu agent correctly ingested the metrics.

If you add the debug handler and configure the `collect-metrics` check to use it, the metrics event printed to the debug-event.json file will be similar to this example:

```json
{
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
```

```
          "10.0.2.15/24",
          "fe80::2a24:13f4:6b:c0b8/64"
        ]
      },
      {
        "name": "eth1",
        "mac": "08:00:27:3d:ce:39",
        "addresses": [
          "172.28.128.63/24",
          "fe80::a00:27ff:fe3d:ce39/64"
        ]
      }
    ]
  },
  "arch": "amd64",
  "libc_type": "glibc",
  "vm_system": "vbox",
  "vm_role": "guest",
  "cloud_provider": "",
  "processes": null
},
"subscriptions": [
  "entity:sensu-centos",
  "linux"
],
"last_seen": 1625000586,
"deregister": false,
"deregistration": {},
"user": "agent",
"redact": [
  "password",
  "passwd",
  "pass",
  "api_key",
  "api_token",
  "access_key",
  "secret_key",
  "private_key",
  "secret"
],
"metadata": {
  "name": "sensu-centos",
```

```json
      "namespace": "default"
    },
    "sensu_agent_version": "6.4.0"
  },
  "check": {
    "command": "metrics-disk-usage.rb",
    "handlers": [
      "debug"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-disk-checks",
      "sensu/sensu-ruby-runtime"
    ],
    "subscriptions": [
      "linux"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 0.060007931,
    "executed": 1625000586,
    "history": [
      {
        "status": 0,
        "executed": 1625000346
      },
      {
        "status": 0,
        "executed": 1625000406
      },
      {
        "status": 0,
        "executed": 1625000466
      },
```

```json
        {
          "status": 0,
          "executed": 1625000526
        },
        {
          "status": 0,
          "executed": 1625000586
        }
      ],
      "issued": 1625000586,
      "output": "sensu-centos.disk_usage.root.used 1515 1625000586\nsensu-
centos.disk_usage.root.avail 40433 1625000586\nsensu-
centos.disk_usage.root.used_percentage 4 1625000586\nsensu-
centos.disk_usage.root.dev.used 0 1625000586\nsensu-centos.disk_usage.root.dev.avail
485 1625000586\nsensu-centos.disk_usage.root.dev.used_percentage 0
1625000586\nsensu-centos.disk_usage.root.run.used 51 1625000586\nsensu-
centos.disk_usage.root.run.avail 446 1625000586\nsensu-
centos.disk_usage.root.run.used_percentage 11 1625000586\nsensu-
centos.disk_usage.root.boot.used 130 1625000586\nsensu-
centos.disk_usage.root.boot.avail 885 1625000586\nsensu-
centos.disk_usage.root.boot.used_percentage 13 1625000586\nsensu-
centos.disk_usage.root.home.used 33 1625000586\nsensu-
centos.disk_usage.root.home.avail 20446 1625000586\nsensu-
centos.disk_usage.root.home.used_percentage 1 1625000586\nsensu-
centos.disk_usage.root.vagrant.used 79699 1625000586\nsensu-
centos.disk_usage.root.vagrant.avail 874206 1625000586\nsensu-
centos.disk_usage.root.vagrant.used_percentage 9 1625000586\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1625000586,
      "occurrences": 5,
      "occurrences_watermark": 5,
      "output_metric_format": "graphite_plaintext",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "collect-metrics",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
```

```
      "scheduler": "memory"
    },
    "metrics": {
      "handlers": null,
      "points": [
        {
          "name": "sensu-centos.disk_usage.root.used",
          "value": 1515,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.avail",
          "value": 40433,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.used_percentage",
          "value": 4,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.dev.used",
          "value": 0,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.dev.avail",
          "value": 485,
          "timestamp": 1625000586,
          "tags": null
        },
        {
          "name": "sensu-centos.disk_usage.root.dev.used_percentage",
          "value": 0,
          "timestamp": 1625000586,
          "tags": null
        },
        {
```

```json
    "name": "sensu-centos.disk_usage.root.run.used",
    "value": 51,
    "timestamp": 1625000586,
    "tags": null
  },
  {
    "name": "sensu-centos.disk_usage.root.run.avail",
    "value": 446,
    "timestamp": 1625000586,
    "tags": null
  },
  {
    "name": "sensu-centos.disk_usage.root.run.used_percentage",
    "value": 11,
    "timestamp": 1625000586,
    "tags": null
  },
  {
    "name": "sensu-centos.disk_usage.root.boot.used",
    "value": 130,
    "timestamp": 1625000586,
    "tags": null
  },
  {
    "name": "sensu-centos.disk_usage.root.boot.avail",
    "value": 885,
    "timestamp": 1625000586,
    "tags": null
  },
  {
    "name": "sensu-centos.disk_usage.root.boot.used_percentage",
    "value": 13,
    "timestamp": 1625000586,
    "tags": null
  },
  {
    "name": "sensu-centos.disk_usage.root.home.used",
    "value": 33,
    "timestamp": 1625000586,
    "tags": null
  },
  {
```

```json
        "name": "sensu-centos.disk_usage.root.home.avail",
        "value": 20446,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.home.used_percentage",
        "value": 1,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.vagrant.used",
        "value": 79699,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.vagrant.avail",
        "value": 874206,
        "timestamp": 1625000586,
        "tags": null
      },
      {
        "name": "sensu-centos.disk_usage.root.vagrant.used_percentage",
        "value": 9,
        "timestamp": 1625000586,
        "tags": null
      }
    ]
  },
  "metadata": {
    "namespace": "default"
  },
  "id": "7468a597-bc3c-4ea7-899c-51c4d2992689",
  "sequence": 5,
  "timestamp": 1625000586
}
```

Next step: Send metrics to a handler

Now that you know how to extract metrics from check output, learn to use a metrics handler to populate service and time-series metrics in InfluxDB. For a turnkey experience with the Sensu InfluxDB Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in InfluxDB.

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.

Install and configure Prometheus

Download and extract Prometheus with these commands:

```
wget https://github.com/prometheus/prometheus/releases/download/v2.6.0/prometheus-
2.6.0.linux-amd64.tar.gz
```

```
tar xvfz prometheus-*.tar.gz
```

```
cd prometheus-*
```

Replace the default `prometheus.yml` configuration file with the following configuration:

```
global:
  scrape_interval: 15s
  external_labels:
    monitor: 'codelab-monitor'

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
```

Start Prometheus in the background:

```
nohup ./prometheus --config.file=prometheus.yml > prometheus.log 2>&1 &
```

Ensure Prometheus is running:

```
ps -ef | grep "[p]rometheus"
```

The response should be similar to this example:

```
vagrant   7647  3937  2 22:23 pts/0    00:00:00 ./prometheus --
config.file=prometheus.yml
```

Install and configure Sensu

Follow the RHEL/CentOS install instructions for the Sensu backend, the Sensu agent, and sensuctl.

Use sensuctl to add an `app_tier` subscription to the entity the Sensu agent is observing.Before you run the following code, Replace `<entity_name>` with the name of the entity on your system.

```
sensuctl entity update <entity_name>
```

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

Run this command to confirm both 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 with these commands:

```
influx -execute "CREATE DATABASE sensu"
```

```
influx -execute "CREATE USER sensu WITH PASSWORD 'sensu'"
```

```
influx -execute "GRANT ALL ON sensu TO sensu"
```

Install and configure Grafana

Install Grafana:

```
sudo yum install -y https://s3-us-west-2.amazonaws.com/grafana-
releases/release/grafana-5.1.4-1.x86_64.rpm
```

Change Grafana's listen port so that it does not conflict with the Sensu web UI:

```
sudo sed -i 's/^;http_port = 3000/http_port = 4000/' /etc/grafana/grafana.ini
```

Create a `/etc/grafana/provisioning/datasources/influxdb.yaml` file, and add an InfluxDB data source:

```

```

```
apiVersion: 1

deleteDatasources:
  - name: InfluxDB
    orgId: 1

datasources:
  - name: InfluxDB
    type: influxdb
    access: proxy
    orgId: 1
    database: sensu
    user: grafana
    password: grafana
    url: http://localhost:8086
```

Start Grafana:

```
sudo systemctl start grafana-server
```

Create a Sensu InfluxDB pipeline

Add the Sensu InfluxDB handler asset

To add the Sensu InfluxDB Handler dynamic runtime asset to Sensu, run the following command:

SHELL

```
cat << EOF | sensuctl create
---
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
    EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "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"
  }
}
EOF
```

Add the Sensu handler

To add the handler definition that uses the Sensu InfluxDB Handler dynamic runtime asset, run:

SHELL

```
cat << EOF | sensuctl create
---
type: Handler
api_version: core/v2
metadata:
  name: influxdb
```

```
  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
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "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"
    ]
  }
}
EOF
```

> **PRO TIP**: `sensuctl create --file` also accepts files that contain multiple resources' definitions.
> You could save both the asset and handler definitions in a single file and use `sensuctl create --file FILE_NAME.EXT` to add them.

Collect Prometheus metrics with Sensu

Add the Sensu Prometheus Collector asset

To add the Sensu Prometheus Collector dynamic runtime asset to Sensu, run the following command:

SHELL

```
cat << EOF | sensuctl create
---
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
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "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"
  }
```

```
  }
  EOF
```

Add a Sensu check to complete the pipeline

To add the <u>check</u> definition that uses the Sensu Prometheus Collector dynamic runtime asset, run:

SHELL

```
cat << EOF | sensuctl create
---
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: []
  interval: 10
  publish: true
  output_metric_format: influxdb_line
  output_metric_handlers:
  - influxdb
  subscriptions:
  - app_tier
  timeout: 0
  runtime_assets:
  - sensu-prometheus-collector
  EOF
```

SHELL

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

```
      "namespace": "default"
    },
    "spec": {
      "command": "sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
  query up",
      "handlers": [],
      "interval": 10,
      "publish": true,
      "output_metric_format": "influxdb_line",
      "output_metric_handlers": [
        "influxdb"
      ],
      "subscriptions": [
        "app_tier"
      ],
      "timeout": 0,
      "runtime_assets": [
        "sensu-prometheus-collector"
      ]
    }
  }
  EOF
```

The check subscription matches the subscription you added to your entity during set-up. The Sensu backend will coordinate check execution for you by comparing the subscriptions in your checks and entities. Sensu automatically executes a check when the check definition includes a subscription that matches a subscription for a Sensu entity.

Open the Sensu web UI to view the events generated by the `prometheus_metrics` check. Visit http://127.0.0.1:3000, and log in as the admin user (created during the initialization step when you installed the Sensu backend).

You can also view the metric event data using sensuctl. Run:

```
sensuctl event list
```

The response should be similar to this example:

Entity	Check	Output	Status	Silenced	Timestamp
_____	_____				
_____					_____
_____	_____				
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. The page should include a graph with initial metrics, similar to:
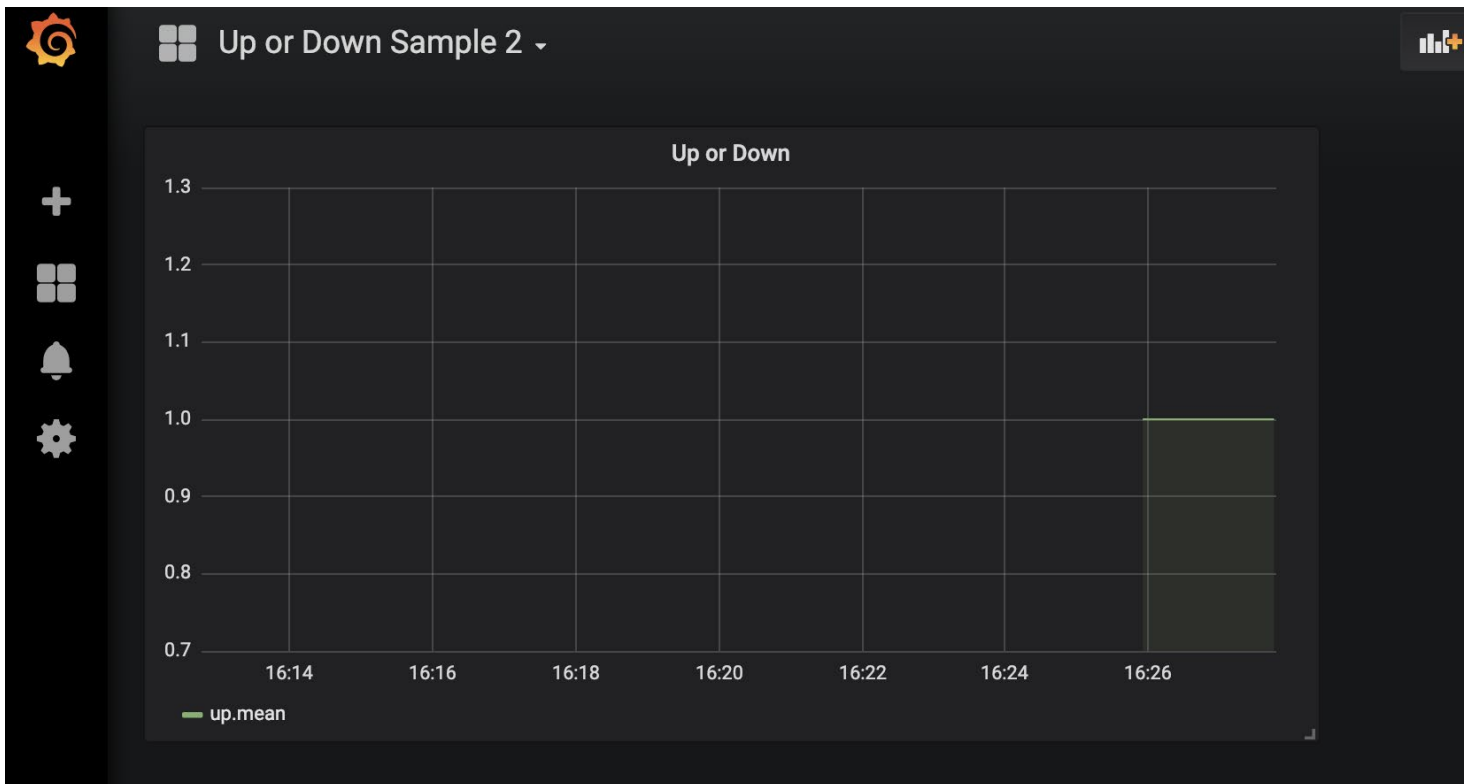
Next steps

You should now have a working observability pipeline with Prometheus scraping metrics. In this pipeline, Sensu Prometheus Collector dynamic runtime asset runs via the `prometheus_metrics` Sensu check and collects metrics from the Prometheus API. The `influxdb` handler sends the metrics to InfluxDB, and you can visualize the metrics in a Grafana dashboard.

Add the Sensu Prometheus Collector to your Sensu ecosystem and include it in your monitoring as code repository. Use Prometheus to gather metrics and use Sensu to send them to the proper final destination. Prometheus has a comprehensive list of additional exporters to pull in metrics.

Hooks reference

Hooks are reusable commands the agent executes in response to a check result before creating an observability event.You can create, manage, and reuse hooks independently of checks.Hooks enrich observability event context by gathering relevant information based on the exit status code of a check (ex: `1`).Hook commands can also receive JSON serialized Sensu client data via `STDIN` .

Hook example

You can use hooks to automate data gathering for incident triage.This example demonstrates a check hook to capture the process tree when a process is not running:

YML

```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

```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 response types

Each **type** of response (ex: `non-zero`) can contain one or more hooks and correspond to one or more exit status codes.Hooks are executed in order of precedence, based on their type:

1. `1` to `255`
2. `ok`
3. `warning`
4. `critical`
5. `unknown`
6. `non-zero`

You can assign one or more hooks to a check in the check definition.review the check specification to configure the `check_hooks` attribute.

Check hooks

Sensu captures the hook command output, status, executed timestamp, and duration and publishes them in the resulting event.

You can use `sensuctl` to view hook command data:

SHELL

```
sensuctl event info entity_name check_name --format yaml
```

SHELL

```
sensuctl event info entity_name check_name --format wrapped-json
```

YML

```yaml
---
type: Event
api_version: core/v2
metadata:
  namespace: default
spec:
  check:
    ...
    hooks:
    - command: df -hT / | grep '/'
      duration: 0.002904412
      executed: 1559948435
      issued: 0
      metadata:
        name: root_disk
        namespace: default
      output: "/dev/mapper/centos-root xfs    41G  1.6G   40G   4% /\n"
      status: 0
      stdin: false
      timeout: 60
```

JSON

```json
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "...": "...",
      "hooks": [
        {
          "command": "df -hT / | grep '/'",
          "duration": 0.002904412,
          "executed": 1559948435,
```

```
          "issued": 0,
          "metadata": {
            "name": "root_disk",
            "namespace": "default"
          },
          "output": "/dev/mapper/centos-root xfs     41G  1.6G   40G   4% /\n",
          "status": 0,
          "stdin": false,
          "timeout": 60
        }
      ]
    }
  }
}
```

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
	YML
| example | ```
type: HookConfig
``` |
| | **JSON** |
| | ```
{
  "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

YML

example	

```yaml
api_version: core/v2
```

JSON

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

metadata

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

YML

example	

```yaml
metadata:
  name: process_tree
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    slack-channel: "#monitoring"
```

JSON

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

spec	
description	Top-level map that includes the hook spec attributes.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
	YML
| example | ```yaml
spec:
  command: ps aux
  timeout: 60
``` |

```
      stdin: false
```

JSON

```json
{
  "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	**YML**
	
	`name: process_tree`
	
	JSON
	
	`{`
	`  "name": "process_tree"`
	`}`

namespace

description	The Sensu RBAC namespace that this hook belongs to.
required	false
type	String
default	`default`

example

YML

```yml
namespace: production
```

JSON

```json
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the hook or last updated the hook. Sensu automatically populates the `created_by` field when the hook is created or updated.
required	false
type	String

example

YML

```yml
created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation data in events that you can use for response and web UI view filtering.
	If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.
	Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

YML

```yml
labels:
  environment: development
  region: us-west-2
```

JSON

```json
{
  "labels": {
    "environment": "development",
    "region": "us-west-2"
  }
}
```

annotations

description	Non-identifying metadata to include with observation data in events that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example	**YML**
	
	``yaml
	annotations:
	  managed-by: ops
	  playbook: www.example.url
	``
	
	JSON
	
	``json
	{
	  "annotations": {
	    "managed-by": "ops",
	    "playbook": "www.example.url"
	  }
	}
	``

Spec attributes

command	
description	Hook command to be executed.
required	true
type	String

YML

example	
| | ```
command: sudo /etc/init.d/nginx start
``` |
| | **JSON** |
| | ```
{
  "command": "sudo /etc/init.d/nginx start"
}
``` |

timeout	
description	Hook execution duration timeout (hard stop). In seconds.
required	false
type	Integer
default	60
	YML
| example | ```
timeout: 30
``` |
| | **JSON** |
| | ```
{
  "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 | **YML**<br><br>```yml
stdin: true
```<br><br>**JSON**<br><br>```json
{
  "stdin": true
}
``` |

runtime_assets

description	Array of Sensu dynamic runtime assets (by their names) required at runtime for execution of the `command`.
required	false
type	Array
| example | **YML**<br><br>```yml
runtime_assets:
- log-context
```<br><br>**JSON**<br><br>```json
{
  "runtime_assets": [
    "log-context"
  ]
}
``` |

Hook for rudimentary auto-remediation

You can use hooks for rudimentary auto-remediation tasks, such as starting a process that is no longer running.

> **NOTE**: Use caution with this approach. Hooks used for auto-remediation will run without regard to the number of event occurrences.

YML

```yaml
---
type: HookConfig
api_version: core/v2
metadata:
  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
  }
}
```

Hook that uses token substitution

You can create check hooks that use token substitution so you can fine-tune check attributes on a per-entity level and re-use the check definition.

> **NOTE**: Token substitution uses entity-scoped metadata, so make sure to set labels at the entity level.

YML

```
---
type: HookConfig
api_version: core/v2
metadata:
  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": {
      "foo": "bar"
    },
```

```
        "name": "tokensub",
        "namespace": "default"
    },
    "spec": {
        "command": "tokensub {{ .labels.foo }}",
        "stdin": false,
        "timeout": 60
    }
}
```

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 (for example, `1`).

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

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).

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
```

To confirm that the hook was added, run:

SHELL

```
sensuctl hook info process_tree --format yaml
```

SHELL

```
sensuctl hook info process_tree --format wrapped-json
```

The response will include the complete hook resource definition in the specified format:

YML

```yaml
---
type: HookConfig
api_version: core/v2
metadata:
  created_by: admin
  name: process_tree
  namespace: default
spec:
  command: ps aux
  runtime_assets: null
  stdin: false
  timeout: 10
```

JSON

```json
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "process_tree",
    "namespace": "default"
  },
  "spec": {
    "command": "ps aux",
    "runtime_assets": null,
    "stdin": false,
    "timeout": 10
  }
}
```

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
```

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.

SHELL

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

SHELL

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

The check hook command result is available in the `hooks` array, within the `check` scope:

YML

```
check:
  ...
  hooks:
  - config:
      name: process_tree
      command: ps aux
      timeout: 10
      namespace: default
    duration: 0.008713605
    executed: 1521724622
    output: ''
    status: 0
```

```
   ...
```

JSON

```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'
```

This example output is truncated for brevity, but it reflects the output of the `ps aux` command specified in the check hook you created:

```
USER        PID %CPU %MEM    VSZ    RSS TTY      STAT START    TIME COMMAND
root          1  0.0  0.3  46164   6704 ?        Ss   Nov17    0:11
```

```
/usr/lib/systemd/systemd --switched-root --system --deserialize 20
root          2  0.0  0.0        0       0 ?         S    Nov17   0:00 [kthreadd]
root          3  0.0  0.0        0       0 ?         S    Nov17   0:01 [ksoftirqd/0]
root          7  0.0  0.0        0       0 ?         S    Nov17   0:01 [migration/0]
root          8  0.0  0.0        0       0 ?         S    Nov17   0:00 [rcu_bh]
root          9  0.0  0.0        0       0 ?         S    Nov17   0:34 [rcu_sched]
```

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.

Tokens reference

Tokens are placeholders in a check definition that the agent replaces with entity information before executing the check. You can use tokens to fine-tune check attributes (like alert thresholds) on a per-entity level while reusing the check definition.

When a check is scheduled to be executed by an agent, it first goes through a token substitution step. The agent replaces any tokens with matching attributes from the entity definition, and then the check is executed. Invalid templates or unmatched tokens return an error, which is logged and sent to the Sensu backend message transport. Checks with token-matching errors are not executed.

Token substitution is supported for check, hook, and dynamic runtime asset definitions. Only entity attributes are available for substitution. Token substitution is not available for event filters because filters already have access to the entity.

Available entity attributes will always have string values, such as labels and annotations.

Example: Token substitution for check thresholds

This example demonstrates a reusable disk usage check. The check command includes `-w` (warning) and `-c` (critical) arguments with default values for the thresholds (as percentages) for generating warning or critical events. The check will compare every subscribed entity's disk space against the default threshold values to determine whether to generate a warning or critical event.

However, the check command also includes token substitution, which means you can add entity labels that correspond to the check commnand tokens to specify different warning and critical values for individual entities. Instead of creating a different check for every set of thresholds, you can use the same check to apply the defaults in most cases and the token-substituted values for specific entities.

Follow this example to set up a reusable check for disk usage:

1. Add the Sensu disk usage check dynamic runtime asset, which includes the command you will need for your check:

   ```
   sensuctl asset add sensu/check-disk-usage:0.4.1
   ```

You will receive a response to confirm that the asset was added:

```
fetching bonsai asset: sensu/check-disk-usage:0.4.1
added asset: sensu/check-disk-usage:0.4.1

You have successfully added the Sensu asset resource, but the asset will not
get downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset
to the appropriate
resource, populate the "runtime_assets" field with ["sensu/check-disk-usage].
```

2. Create the `check-disk-usage` check:

SHELL

```
cat << EOF | sensuctl create
---
type: CheckConfig
api_version: core/v2
metadata:
  name: check-disk-usage
  namespace: default
spec:
  check_hooks: []
  command: check-disk-usage -w {{index .labels "disk_warning" | default 80}} -
c
    {{.labels.disk_critical | default 90}}
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  output_metric_tags: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - sensu/check-disk-usage
```

```
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    ttl: 0
  EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-disk-usage",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": [],
    "command": "check-disk-usage -w {{index .labels "disk_warning" | default
80}} -c {{.labels.disk_critical | default 90}}",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "output_metric_tags": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu/check-disk-usage"
    ],
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
```

```
      "ttl": 0
    }
  }
  EOF
```

This check will run on every entity with the subscription `system` .According to the default values in the command, the check will generate a warning event at 80% disk usage and a critical event at 90% disk usage.

3. To receive alerts at different thresholds for an existing entity with the `system` subscription, add `disk_warning` and `disk_critical` labels to the entity.

   Use sensuctl to open an existing entity in a text editor:

   ```
   sensuctl edit entity ENTITY_NAME
   ```

   And add the following labels in the entity metadata:

   ```
   labels:
     disk_warning: "65"
     disk_critical: "75"
   ```

After you save your changes, the `check-disk-usage` check will substitute the `disk_warning` and `disk_critical` label values to generate events at 65% and 75% of disk usage, respectively, for this entity only.The check will continue to use the 80% and 90% default values for other subscribed entities.

Add a hook that uses token substitution

Now you have a resusable check that will send disk usage alerts at default or entity-specific thresholds. You may want to add a hook to list more details about disk usage for warning and critical events.

The hook in this example will list disk usage in human-readable format, with error messages filtered from the hook output.By default, the hook will list details for the top directory and the first layer of subdirectories.As with the `check-disk-usage` check, you can add a `disk_usage_root` label to individual entities to specify a different directory for the hook via token substitution.

1. Add the hook definition:

SHELL

```
cat << EOF | sensuctl create
---
type: HookConfig
api_version: core/v2
metadata:
  name: disk_usage_details
  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
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "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
  }
}
EOF
```

2. Add the hook to the `check-disk-usage` check.

Use sensuctl to open the check in a text editor:

```
sensuctl edit check check-disk-usage
```

Update the check definition to include the `disk_usage_details` hook for `non-zero` events:

```
check_hooks:
- non-zero:
  - disk_usage_details
```

3. As with the disk usage check command, the hook command includes a token substitution option To use a specific directory instead of the default for specific entities, edit the entity definition to add a `disk_usage_root` label and specify the directory:

Use sensuctl to open the entity in a text editor:

```
sensuctl edit entity ENTITY_NAME
```

Add the `disk_usage_root` label with the desired substitute directory in the entity metadata:

```
labels:
  disk_usage_root: "/substitute-directory"
```

After you save your changes, for this entity, the hook will substitute the directory you specified for the `disk_usage_root` label to provide additional disk usage details for every non-zero event the `check-disk-usage` check generates.

Manage entity labels

You can use token substitution with any defined entity attributes, including custom labels. read the entity reference for information about managing entity labels for proxy entities and agent entities.

Manage dynamic runtime assets

You can use token substitution in the URLs of your your <u>dynamic runtime asset</u> definitions.Token substitution allows you to host your dynamic runtime assets at different URLs (such as at different datacenters) without duplicating your assets, as shown in the following example:

YML

```yml
type: Asset
api_version: core/v2
metadata:
  name: sensu-go-hello-world
  namespace: default
spec:
  builds:
  - sha512:
07665fda5b7c75e15e4322820aa7ddb791cc9338e38444e976e601bc7d7970592e806a7b88733690a238
b7325437d31f85e98ae2fe47b008ca09c86530da9600
    url: "{{ .labels.asset_url }}/sensu-go-hello-world-0.0.1.tar.gz"
```

JSON

```json
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-go-hello-world",
    "namespace": "default"
  },
  "spec": {
    "builds": [
      {
        "sha512":
"07665fda5b7c75e15e4322820aa7ddb791cc9338e38444e976e601bc7d7970592e806a7b88733690a23
8b7325437d31f85e98ae2fe47b008ca09c86530da9600",
        "url": "{{ .labels.asset_url }}/sensu-go-hello-world-0.0.1.tar.gz"
      }
    ]
  }
}
```

With this asset definition, which includes the `.labels.asset_url` token substitution, checks and hooks can include `sensu-go-hello-world` as a dynamic runtime assets and Sensu Go will use the token substitution for the agent's entity.Handlers and mutators can also include `sensu-go-hello-world` as a dynamic runtime asset, but Sensu Go will use the token subtitution for the backend's entity instead of the agent's entity.

You can also use token substitution to customize dynamic runtime asset headers (for example, to include secure information for authentication).

> **NOTE**: To maintain security, you cannot use token substitution for a dynamic runtime asset's SHA512 value.

Token specification

Sensu Go uses the Go template package to implement token substitution.Use double curly braces around the token and a dot before the attribute to be substituted: `{{ .system.hostname }}`.

Token substitution syntax

Tokens are invoked by wrapping references to entity attributes and labels with double curly braces, such as `{{ .name }}` to substitute an entity's name.Access nested Sensu entity attributes with dot notation (for example, `system.arch`).

- `{{ .name }}` would be replaced with the entity `name` attribute

- `{{ .labels.url }}` would be replaced with a custom label called `url`

- `{{ .labels.disk_warning }}` would be replaced with a custom label called `disk_warning`

- `{{ index .labels "disk_warning" }}` would be replaced with a custom label called `disk_warning`

- `{{ index .labels "cpu.threshold" }}` would be replaced with a custom label called `cpu.threshold`

> **NOTE**: When an annotation or label name has a dot (for example, `cpu.threshold`), you must use the template index function syntax to ensure correct processing because the dot notation is also used for object nesting.

Token substitution default values

If an attribute is not provided by the <u>entity</u>, a token's default value will be substituted. Token default values are separated by a pipe character and the word "default" (`| default`). Use token default values to provide a fallback value for entities that are missing a specified token attribute.

For example, `{{.labels.url | default "https://sensu.io"}}` would be replaced with a custom label called `url`. If no such attribute called `url` is included in the entity definition, the default (or fallback) value of `https://sensu.io` will be used to substitute the token.

Token substitution with quoted strings

You can escape quotes to express quoted strings in token substitution templates as shown in the <u>Go template package examples</u>. For example, to provide `"substitution"` as a default value for entities that are missing the `website` attribute (including the quotation marks):

```
{{ .labels.website | default "\"substitution\"" }}
```

Unmatched tokens

If a token is unmatched during check preparation, the agent check handler will return an error, and the check will not be executed. Unmatched token errors are similar to this example:

```
error: unmatched token: template: :1:22: executing "" at <.system.hostname>: map has
no entry for key "System"
```

Check config token errors are logged by the agent and sent to Sensu backend message transport as check failures.

Token data type limitations

As part of the substitution process, Sensu converts all tokens to strings. This means that token

substitution cannot be applied to any non-string values like numbers or Booleans, although it can be applied to strings that are nested inside objects and arrays.

For example, token substitution **cannot** be used for specifying a check interval because the interval attribute requires an *integer* value. Token substitution **can** be used for alerting thresholds because thos values are included within the command *string*.

Filter your observation data

Sensu Observability Pipeline
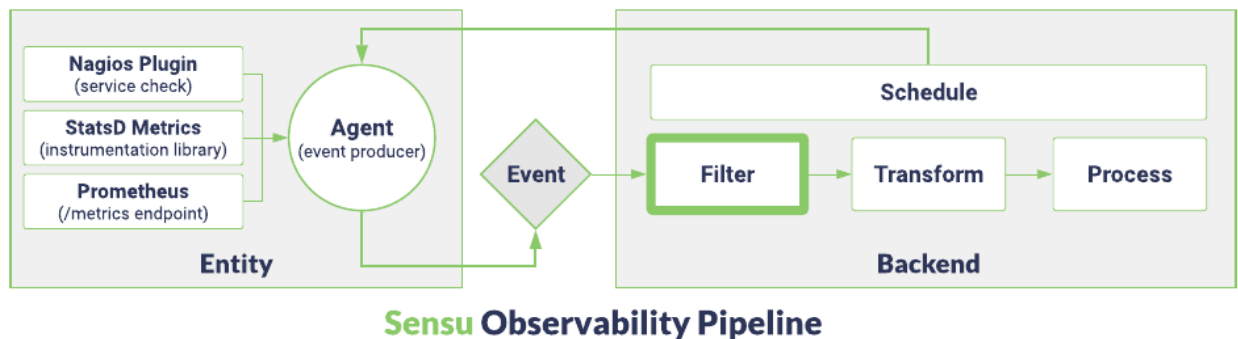
In the filter stage, Sensu executes <u>event filters</u>.

The filter stage of the Sensu observability pipeline applies the conditions, triggers, and thresholds you specify in your <u>event filter</u> defintions to the events your checks generate.Event filters give you control over which events continue through your pipeline and become alerts.For example, use the <u>built-in is_incident event filter</u> to allow only high-priority events through your Sensu pipeline and reduce noise for operators.

To tell Sensu which event filters you want to apply, you list them in your <u>handler</u> definition.Sensu compares your observation data in events against the <u>expressions</u> in your event filters to determine whether each event should continue through the pipeline or be removed.Event filters can be <u>inclusive or exclusive</u>, so you can require events to match or not match your filter expressions.

Here's an example that shows the resource definition for an event filter that would allow handling for only events with the custom entity label `"region": "us-west-1"`:

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: production_filter
  namespace: default
spec:
  action: allow
  expressions:
  - event.entity.labels['region'] == 'us-west-1'
```

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "production_filter",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.entity.labels['region'] == 'us-west-1'"
    ]
  }
}
```

Sensu applies event filters in the order that they are listed in your handler definition. Any events that the filters do not remove from your pipeline will be processed according to your handler configuration.

As soon as an event filter removes an event from your pipeline because it does not meet the conditions, triggers, or thresholds you specified, the Sensu observability pipeline ceases analysis for the event. Sensu will not transform or process events that your event filter removes from your pipeline.

Use Bonsai, the Sensu asset hub, to discover, download, and share Sensu event filter dynamic runtime assets. Read Use assets to install plugins to get started.

Event filter reference

Sensu executes event filters during the **filter** stage of the observability pipeline.

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 observation data in events.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.

Event filter example (minimum required attributes)

This example shows the minimum required attributes for an event filter resource:

YML

```
type: EventFilter
api_version: core/v2
metadata:
  name: filter_minimum
```

```
    namespace: default
  spec:
    action: allow
    expressions:
    - event.check.occurrences == 1
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "filter_minimum",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ]
  }
}
```

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 (regex) specified in ECMAScript 5 will work. Review the Otto README for more details.

Use Go regex syntax to create event filter expressions that combine any available event, check, or entity attributes with `match(<regex>)` .

For example, this event filter allows handling for events whose `event.check.name` ends with `metrics` :

YML

```
---
type: EventFilter
api_version: core/v2
metadata:
  name: metrics-checks-only
  namespace: default
spec:
  action: allow
  expressions:
  - event.check.name.match(/metrics$/)
```

JSON

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

```json
      "metadata": {
          "name": "metrics-checks-only",
          "namespace": "default"
      },
      "spec": {
          "action": "allow",
          "expressions": [
              "event.check.name.match(/metrics$/)"
          ]
      }
  }
```

Here's another example that uses regex matching for event entity labels. This event filter allows handling for events created by entities with the `region` label `us-west-1`, `us-west-2`, or `us-west-3`:

YML

```yaml
---
type: EventFilter
api_version: core/v2
metadata:
  name: us-west-events
  namespace: default
spec:
  action: allow
  expressions:
  - event.entity.labels.region.match(/us-west-\b[1-3]\b/)
```

JSON

```json
{
    "type": "EventFilter",
    "api_version": "core/v2",
    "metadata": {
        "name": "us-west-events",
        "namespace": "default"
    },
    "spec": {
        "action": "allow",
```

```
      "expressions": [
          "event.entity.labels.region.match(/us-west-\b[1-3]\b/)"
      ]
    }
  }
```

Filter dynamic runtime assets

Sensu event filters can have dynamic runtime assets that are included in their execution context.When valid dynamic runtime assets are associated with an event filter, Sensu evaluates any files it finds that have a `.js` extension before executing the filter.The result of evaluating the scripts is cached for a given asset set for the sake of performance.For an example of how to implement an event filter as an asset, read Reduce alert fatigue.

Built-in event filters

Sensu includes built-in event filters to help you customize event pipelines for metrics and alerts.To start using built-in event filters, read Send Slack alerts and Plan maintenance.

> **NOTE**: Sensu Go does not include the built-in occurrence-based event filter in Sensu Core 1.x, but you can replicate its functionality with *the repeated events filter definition*.

Built-in filter: is_incident

The is_incident event filter is included in every installation of the Sensu backend.You can use the is_incident filter to allow only high-priority events through a Sensu pipeline.For example, you can use the is_incident filter to reduce noise when sending notifications to Slack.When applied to a handler, the is_incident filter allows warning (`"status": 1`), critical (`"status": 2`), other (unknown or custom status), and resolution events to be processed.

To use the is_incident event filter, include `is_incident` in the handler configuration `filters` array:

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
    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
XXXXXXXXXXX"
    ],
    "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 (unknown or custom status)	✅	
resolution event such as 1 –> 0 or 3 –> 0	✅	

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

```
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

```yaml
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 with Sensu query expressions

You can write custom event filter expressions as Sensu query expressions using the event data attributes described in this section. For more information about event attributes, read the event reference.

Syntax quick reference

operator	description
`===` / `!==`	Identity operator / Nonidentity operator
`==` / `!=`	Equality operator / Inequality operator
`&&` / `||`	Logical AND / Logical OR
`<` / `>`	Less than / Greater than
`<=` / `>=`	Less than or equal to / Greater than or equal to

Event attributes available to filters

attribute	type	description
`event.has_check`	Boolean	Returns true if the event contains check data
`event.has_metrics`	Boolean	Returns true if the event contains metrics
`event.is_incident`	Boolean	Returns true for critical alerts (status `2`), warnings (status `1`), and resolution events (status `0` transitioning from status `1` or `2`)
`event.is_resolution`	Boolean	Returns true if the event status is OK (`0`) and the previous event was of a non-zero status
`event.is_silenced`	Boolean	Returns true if the event matches an active silencing entry
`event.timestamp`	integer	Time that the event occurred in seconds since the Unix epoch

Check attributes available to filters

attribute	type	description
`event.check.annotations`	map	Custom annotations applied to the check
`event.check.command`	string	The command executed by the check
`event.check.cron`	string	Check execution schedule using cron syntax
`event.check.discard_output`	Boolean	Whether the check is configured to discard check output from event data
`event.check.duration`	float	Command execution time in seconds
`event.check.env_vars`	array	Environment variables used with command execution

`event.check.execu ted`	integ er	Time that the check was executed in seconds since the Unix epoch
`event.check.handl ers`	array	Sensu event handlers assigned to the check
`event.check.high_ flap_threshold`	integ er	The check's flap detection high threshold in percent state change
`event.check.histo ry`	array	Check status history for the last 21 check executions
`event.check.hook s`	array	Check hook execution data
`event.check.inter val`	integ er	The check execution frequency in seconds
`event.check.issue d`	integ er	Time that the check request was issued in seconds since the Unix epoch
`event.check.label s`	map	Custom labels applied to the check
`event.check.last_ ok`	integ er	The last time that the check returned an OK status (`0`) in seconds since the Unix epoch
`event.check.low_fl ap_threshold`	integ er	The check's flap detection low threshold in percent state change
`event.check.max_o utput_size`	integ er	Maximum size of stored check outputs in bytes
`event.check.name`	strin g	Check name
`event.check.occur rences`	integ er	The number of preceding events with the same status as the current event
`event.check.occur rences_watermark`	integ er	For resolution events, the number of preceding events with a non-OK status
`event.check.outpu t`	strin g	The output from the execution of the check command

`event.check.output_metric_format`	string	The metric format generated by the check command: `nagios_perfdata`, `graphite_plaintext`, `influxdb_line`, `opentsdb_line`, or `prometheus_text`
`event.check.output_metric_handlers`	array	Sensu metric handlers assigned to the check
`event.check.proxy_entity_name`	string	The entity name, used to create a proxy entity for an external resource
`event.check.proxy_requests`	map	Proxy request configuration
`event.check.publish`	Boolean	Whether the check is scheduled automatically
`event.check.round_robin`	Boolean	Whether the check is configured to be executed in a round-robin style
`event.check.runtime_assets`	array	Sensu dynamic runtime assets used by the check
`event.check.state`	string	The state of the check: `passing` (status `0`), `failing` (status other than `0`), or `flapping`
`event.check.status`	integer	Exit status code produced by the check: `0` (OK), `1` (warning), `2` (critical), or other status (unknown or custom status)
`event.check.stdin`	Boolean	Whether the Sensu agent writes JSON-serialized entity and check data to the command process' STDIN
`event.check.subscriptions`	array	Subscriptions that the check belongs to
`event.check.timeout`	integer	The check execution duration timeout in seconds
`event.check.total_state_change`	integer	The total state change percentage for the check's history
`event.check.ttl`	integer	The time-to-live (TTL) until the event is considered stale, in seconds

attribute	type	description
`event.metrics.han dlers`	array	Sensu metric <u>handlers</u> assigned to the check
`event.metrics.poi nts`	array	<u>Metrics data points</u> including a name, timestamp, value, and tags

Entity attributes available to filters

attribute	type	description
`event.entity.anno tations`	map	Custom <u>annotations</u> assigned to the entity
`event.entity.dere gister`	Bool ean	Whether the agent entity should be removed when it stops sending <u>keepalive messages</u>
`event.entity.dere gistration`	map	A map that contains a handler name for use when an entity is deregistered
`event.entity.enti ty_class`	strin g	The entity type: usually `agent` or `proxy`
`event.entity.labe ls`	map	Custom <u>labels</u> assigned to the entity
`event.entity.last _seen`	integ er	Timestamp the entity was last seen in seconds since the Unix epoch
`event.entity.nam e`	strin g	Entity name
`event.entity.reda ct`	array	List of items to redact from log messages
`event.entity.subs criptions`	array	List of subscriptions assigned to the entity
`event.entity.syst em`	map	Information about the <u>entity's system</u>
`event.entity.syst em.arch`	strin g	The entity's system architecture

`event.entity.system.hostname`	string	The entity's hostname
`event.entity.system.network`	map	The entity's network interface list
`event.entity.system.os`	string	The entity's operating system
`event.entity.system.platform`	string	The entity's operating system distribution
`event.entity.system.platform_family`	string	The entity's operating system family
`event.entity.system.platform_version`	string	The entity's operating system version
`event.entity.user`	string	Sensu RBAC username used by the agent entity

Build event filter expressions with JavaScript execution functions

COMMERCIAL FEATURE: Access built-in JavaScript event filter execution functions in the packaged Sensu Go distribution. For more information, read Get started with commercial features.

In addition to Sensu query expressions, Sensu includes several built-in JavaScript functions for event filter execution:

- `sensu.FetchEvent`
- `sensu.CheckStatus`
- `sensu.ListEvents`

Use these functions to query your event stores for other events in the same namespace.

For example, to handle only events for the `server01` entity *and* the `disk` check, use the `sensu.FetchEvent` function in your event filter expressions:

```
"expressions": [
  '(function () { var diskEvent = sensu.FetchEvent("server01", "disk"); if
(diskEvent == nil) { return false; } return diskEvent.check.status == 0; }) ()'
]
```

`sensu.EventStatus`

The `sensu.EventStatus` function takes zero or more checks as arguments. It returns an array of status codes for the events associated with the specified checks.

If you do not specify any checks, the function always returns an empty array.

You can refer to the checks as strings:

```
sensu.EventStatus("database", "disk")
```

If you pass the check names as strings, Sensu assumes that the entities are the same as those in the events being filtered.

You can also refer to the checks in objects that include both the entity and check name. For example:

```
sensu.EventStatus({entity: "server01", check: "disk"}, {entity: "server01", check:
"database"})
```

In both cases, if no event matches the specified entities and checks, Sensu will raise an error.

`sensu.FetchEvent`

The `sensu.FetchEvent` function loads the Sensu event that corresponds to the specified entity and check names.

The format is `sensu.FetchEvent(entity, check)`. For example:

```
sensu.FetchEvent("server01", "disk")
```

You can only load events from the same namespace as the event being filtered. The returned object uses the same format as responses for the events API.

If an event does not exist for the specified entity and check names, Sensu will raise an error.

`sensu.ListEvents`

The `sensu.ListEvents` function returns an array of all events in the same namespace as the event being filtered.

> **NOTE**: *If you have many events in the namespace, this function may require a substantial amount of time to return them.*

For example:

```
sensu.ListEvents()
```

The events in the returned array use the same format as responses for the events API.

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
	YML
example	

```yml
type: EventFilter
```

JSON

```json
{
  "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
	YML
example	

```yml
api_version: core/v2
```

JSON

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

metadata

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

```yaml
metadata:
  name: filter-weekdays-only
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    slack-channel: "#monitoring"
```

JSON

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

spec

description	Top-level map that includes the event filter <u>spec attributes</u>.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	**YML**

```yml
spec:
  action: allow
  expressions:
  - event.entity.namespace == 'production'
  runtime_assets: []
```

JSON

```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

YML

example
```
name: filter-weekdays-only
```

JSON

```
{
  "name": "filter-weekdays-only"
}
```

namespace

description	Sensu RBAC namespace that the event filter belongs to.
required	false
type	String
default	`default`

YML

example
```
namespace: production
```

JSON

```
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the filter or last updated the filter. Sensu automatically populates the `created_by` field when the

	filter is created or updated.
required	false
type	String

YML

```yml
created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with event data that you can use for response and web UI view filtering.
	If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.
	Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

YML

example	

```yml
labels:
  environment: development
```

```
    region: us-west-2
```

```json
{
  "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 web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

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

```
    }
```

Spec attributes

action	
description	Action to take with the event if the event filter expressions match. Read Inclusive and exclusive event filters for more information.
required	true
type	String
allowed values	`allow` , `deny`

example

YML

```yml
  action: allow
```

JSON

```json
  {
    "action": "allow"
  }
```

expressions	
description	Event filter expressions to be compared with event data. You can reference event metadata without including the `metadata` scope (for example, `event.entity.namespace`).
required	true
type	Array

YML

example	
| | ```yaml
expressions:
- event.check.team == 'ops'
``` |

JSON

```json
{
  "expressions": [
    "event.check.team == 'ops'"
  ]
}
```

runtime_assets

description	Dynamic runtime assets to apply to the event filter's execution context. JavaScript files in the lib directory of the dynamic runtime asset will be evaluated.
required	false
type	Array of string
default	[] YML
| example | ```yaml
runtime_assets:
- underscore
``` |

JSON

```json
{
  "runtime_assets": [
    "underscore"
  ]
}
```

Use JavaScript libraries with Sensu filters

You can include JavaScript libraries in their event filter execution context with <u>dynamic runtime assets</u>. For instance, if you package underscore.js into a Sensu asset, you can use functions from the underscore library for filter expressions:

YML

```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

```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"]
  }
}
```

Filter for production events

The following event filter allows handling for only events with a custom entity label `"environment": "production"`:

YML

```yml
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'"
    ]
  }
}
```

Filter for non-production events

The following event filter discards events with a custom entity label `"environment": "production"`, allowing handling only for events without an `environment` label or events with `environment` set to something other than `production`.

> **NOTE**: `action` is `deny`, so this is an exclusive event filter. If evaluation returns false, the event is handled.

YML

```yml
type: EventFilter
api_version: core/v2
metadata:
  name: not_production
  namespace: default
spec:
  action: deny
  expressions:
  - event.entity.labels['environment'] == 'production'
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "not_production",
    "namespace": "default"
  },
  "spec": {
    "action": "deny",
    "expressions": [
      "event.entity.labels['environment'] == 'production'"
    ]
  }
}
```

Filter for state change only

This example demonstrates how to use the `state_change_only` inclusive event filter to reproduce the behavior of a monitoring system that alerts only on state change:

YML

```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

```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": []
  }
}
```

Filter for repeated events

In this example, the `filter_interval_60_hourly` event filter will match event data with a check `interval` of `60` seconds *AND* an `occurrences` value of `1` (the first occurrence) *OR* any `occurrences` value that is evenly divisible by 60 via a modulo operator calculation (calculating the remainder after dividing `occurrences` by 60):

YML

```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

```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": []
    }
  }
```

Filter to reduce alert fatigue for keepalive events

This example `keepalive_timeouts` event filter will match event data with an occurrences value of 1 OR any occurrences value that matches 15 minutes via a modulo operator calculation. This limits keepalive timeout event alerts to the first occurrence and every 15 minutes thereafter.

This example uses conditional JavaScript logic to check for an entity-level annotation, `keepalive_alert_minutes`, and if it exists, parses the annotation value as an integer. If the annotation does not exist, the event filter uses 15 minutes for the alert cadence.

YML

```yaml
type: EventFilter
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: keepalive_timeouts
  namespace: default
spec:
  action: allow
  expressions:
   - is_incident
   - event.check.occurrences == 1 || event.check.occurrences % parseInt( 60 * (
'keepalive_alert_minutes' in event.entity.annotations ?
parseInt(event.entity.annotations.keepalive_alert_minutes): 15) /
event.check.timeout ) == 0
  runtime_assets: []
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
```

```
      "name": "keepalive_timeouts",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "spec": {
      "action": "allow",
      "expressions": [
        "is_incident",
        "event.check.occurrences == 1 || event.check.occurrences % parseInt( 60 * (
'keepalive_alert_minutes' in event.entity.annotations ?
parseInt(event.entity.annotations.keepalive_alert_minutes): 15) /
event.check.timeout ) == 0"
      ],
      "runtime_assets": []
    }
}
```

Filter for events during office hours only

This event filter evaluates the event timestamp to determine if the event occurred between 9 AM and 5 PM UTC on a weekday.Remember that `action` is equal to `allow`, so this is an inclusive event filter.I evaluation returns false, the event will not be handled.

YML

```
type: EventFilter
api_version: core/v2
metadata:
  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": []
  }
}
```

Disable alerts without a silence

This filter allows you to disable alerts without creating silences.

Add the filter name to the `filters` array for any handler you want to control. To disable alerts, change the filter's `action` attribute value from `allow` to `deny`.

YML

```yml
---
type: EventFilter
api_version: core/v2
metadata:
  name: emergency_alert_control
spec:
  action: allow
  expressions:
  - event.has_check
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "emergency_alert_control"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.has_check"
    ]
  }
}
```

Reduce alert fatigue with event filters

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

Use event filters to customize alert policies, improve contact routing, eliminate notification noise from recurring events, and filter events from systems in pre-production environments.

In this guide, you 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:

- ¬ Use sensuctl

- ¬ Use a filter dynamic runtime asset

Approach 1: Use sensuctl to create an event filter

First, create an event filter called `hourly` that matches new events (where the event's `occurrences` is equal to `1`) or hourly events (every hour after the first occurrence, calculated with the check's `interval` and the event's `occurrences`).

Events in Sensu Go are handled regardless of check execution status.Even successful check events are passed through the pipeline, so you'll need to add a clause for non-zero status.

```
sensuctl filter create hourly \
--action allow \
--expressions "event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0"
```

You should receive a confirmation message:

```
Created
```

This sensuctl command creates an event filter resource with the following definition:

YML

```yaml
---
type: EventFilter
api_version: core/v2
metadata:
  created_by: admin
  name: hourly
  namespace: default
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0
  runtime_assets: null
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "hourly",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1 || event.check.occurrences % (3600 /
event.check.interval) == 0"
    ],
    "runtime_assets": null
  }
}
```

If you want to share and reuse this event filter like code, you can <u>save it to a file</u> and start building a <u>monitoring as code repository</u>.

Assign the event filter to a handler

Now that you've created the `hourly` event filter, you can assign it to a handler. In this case, because you want to reduce the number of Slack messages Sensu sends, you'll apply your `hourly` event filter to the `slack` handler created in <u>Send Slack alerts with handlers</u>. You'll also add the built-in `is_incident` filter so that only failing events are handled.

> **NOTE**: If you haven't already created the `slack` handler, follow <u>Send Slack alerts with handlers</u> before continuing with this step.

To update the handler, run:

```
sensuctl handler update slack
```

Follow the prompts to add the `hourly` and `is_incident` event filters to the `slack handler` :

```
? Environment variables:
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
? Filters: hourly,is_incident
? Mutator:
? Timeout: 0
? Type: pipe
? Runtime Assets: sensu-slack-handler
? Command: sensu-slack-handler --channel '#monitoring'
```

You will receive a confirmation message:

```
Updated
```

To view the updated `slack` handler resource definition:

```shell
sensuctl handler info slack --format yaml
```

SHELL

```shell
sensuctl handler info slack --format wrapped-json
```

The updated handler definition will be similar to this example:

YML

```yaml
---
type: Handler
api_version: core/v2
metadata:
  created_by: admin
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters:
  - hourly
  - is_incident
  handlers: null
  runtime_assets:
  - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

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

```
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX"
    ],
    "filters": [
      "hourly",
      "is_incident"
    ],
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

In addition to using this approach with `sensuctl` to interactively create an event filter, you can create more reusable event filters with dynamic runtime assets. Read on to learn how.

Approach 2: Use an event filter dynamic runtime asset

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

In this approach, the first step is to obtain an event filter dynamic runtime asset that will allow you to replicate the behavior of the `hourly` event filter created in Approach 1 via `sensuctl`.

Use `sensuctl asset add` to register the fatigue check filter dynamic runtime asset:

```
sensuctl asset add sensu/sensu-go-fatigue-check-filter:0.8.1 -r fatigue-filter
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-go-fatigue-check-filter:0.8.1
added asset: sensu/sensu-go-fatigue-check-filter:0.8.1

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["fatigue-filter"].
```

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 hub.

> **NOTE**: *Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read the asset reference for more information about dynamic runtime asset builds.*

You've registered the dynamic runtime asset, but you still need to create the filter.

Create a file named `sensu-fatigue-check-filter.yml` or `sensu-fatigue-check-filter.json` in your Sensu installation to store the event filter definition.Copy this this filter definition into the file and save it:

YML

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

```
  - fatigue-filter
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "fatigue_check",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "fatigue_check(event)"
    ],
    "runtime_assets": [
      "fatigue-filter"
    ]
  }
}
```

Then, use sensuctl to create a filter named `fatigue_check` from the file:

SHELL

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

SHELL

```shell
sensuctl create -f sensu-fatigue-check-filter.json
```

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

Annotate a check for filter dynamic runtime asset use

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

YML

```yml
---
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: null
  output_metric_tags: null
  proxy_entity_name: ''
  publish: true
  round_robin: false
  runtime_assets: null
  stdin: false
  subdue:
  subscriptions:
  - linux
  timeout: 0
  ttl: 0
```

JSON

```json
{
  "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": null,
    "handlers": [
      "email"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "output_metric_tags": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "linux"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

Notice the annotations under the `metadata` scope. The annotations are required for the filter dynamic
runtime asset to work the same way as the interactively created event filter. Specifically, the annotations
in this check definition are doing several things:

1. `fatigue_check/occurrences` : Tells the event filter on which occurrence to send the event for
   further processing

2. `fatigue_check/interval` : Tells the event filter the interval at which to allow additional events to be processed (in seconds)
3. `fatigue_check/allow_resolution` : Determines whether to pass a `resolve` event through to the filter

For more information about configuring these values, read the Sensu Go Fatigue Check Filter README.Next, you'll 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 the `slack` handler to use it:

```
sensuctl handler update slack
```

Follow the prompts to add the `fatigue_check` and `is_incident` event filters to the `slack` handler :

```
? Environment variables:
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
? Filters: fatigue_check,is_incident
? Mutator:
? Timeout: 0
? Type: pipe
? Runtime Assets: sensu-slack-handler
? Command: sensu-slack-handler --channel '#monitoring'
```

You will receive a confirmation message:

```
Updated
```

To view the updated `slack` handler definition:

SHELL

```
sensuctl handler info slack --format yaml
```

```
sensuctl handler info slack --format wrapped-json
```

The updated handler definition will be similar to this example:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  created_by: admin
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters:
  - fatigue_check
  - is_incident
  handlers: null
  runtime_assets:
  - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "slack",
    "namespace": "default"
  },
```

```
    "spec": {
      "command": "sensu-slack-handler --channel '#monitoring'",
      "env_vars": [
        "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX"
      ],
      "filters": [
        "fatigue_check",
        "is_incident"
      ],
      "handlers": null,
      "runtime_assets": [
        "sensu-slack-handler"
      ],
      "secrets": null,
      "timeout": 0,
      "type": "pipe"
    }
  }
```

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 (read Troubleshoot Sensu 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 dynamic runtime asset to help reduce alert fatigue, read the filters reference for in-depth information about event filters.

Route alerts with event filters

Every alert has an ideal first responder: a team or person who knows how to triage and address the issue.Sensu contact routing lets you alert the right people using their preferred contact methods, 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 alerts to the dev team but not to the ops team or the fallback option.
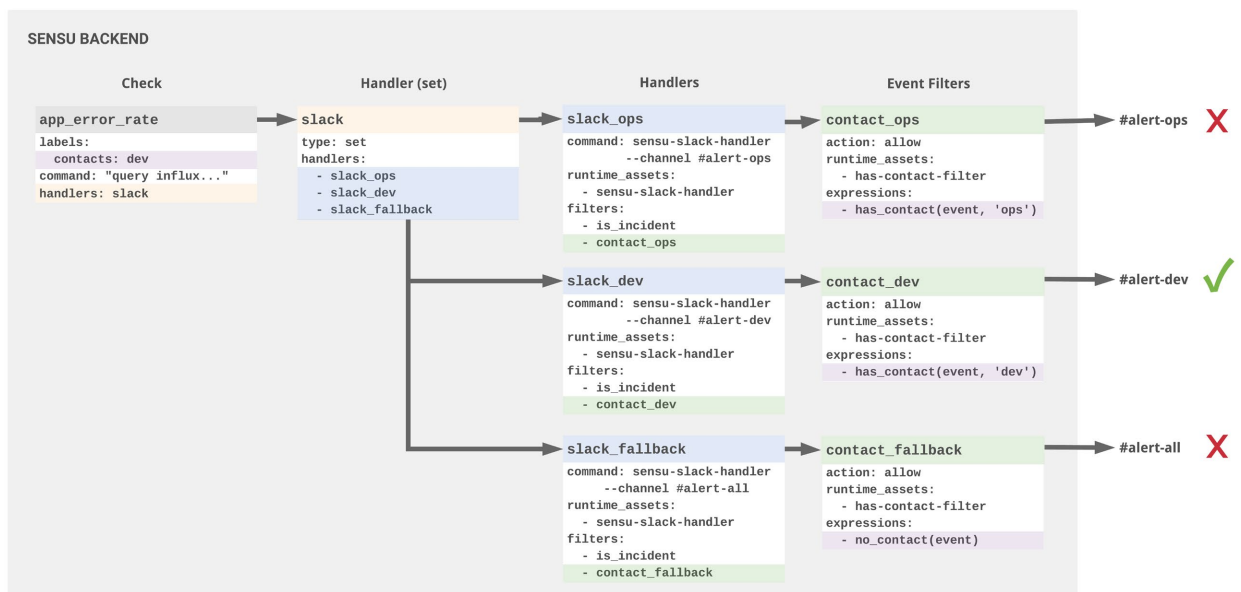
Prerequisites

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 dynamic runtime asset

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

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

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-go-has-contact-filter:0.2.0
added asset: sensu/sensu-go-has-contact-filter:0.2.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["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 dynamic runtime asset definition from Bonsai.

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

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read *the asset reference* for more information about dynamic runtime asset builds.

2. Create contact filters

The Bonsai documentation for the asset explains that the has-contact dynamic runtime asset supports two functions:

- `has_contact`, which takes the Sensu event and the contact name as arguments
- `no_contact`, which is available as a fallback in the absence of contact labels and takes only the event as an argument

You'll use these functions to create event filters that represent the three actions that the Sensu Slack handler can take on an event: contact the ops team, contact the dev team, and contact the fallback option.

event filter name	expression	description
`contact_ops`	`has_contact(event, "ops")`	Allow events with the entity or check label `contacts: ops`
`contact_dev`	`has_contact(event, "dev")`	Allow events with the entity or check label `contacts: dev`
`contact_fallback`	`no_contacts(event)`	Allow events without an entity or check `contacts` label

Use sensuctl to create the three event filters:

TEXT

```
echo '---
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
```

TEXT

```
echo '{
  "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
```

You can also save these event filter resource definitions to a file named `filters.yml` or

`filters.json` in your Sensu installation.When you're ready to manage your observability configurations the same way you do any other code, your `filters.yml` or `filters.json` file can become a part of your <u>monitoring as code</u> repository.

Use sensuctl to confirm that the event filters were added:

```
sensuctl filter list
```

The response should list the new `contact_ops` , `contact_dev` , and `contact_fallback` event filters:

```
    Name       Action      Expressions
 _____ _____ _____

  contact_dev      allow    (has_contact (event, "dev") )
  contact_fallback  allow    (no_contacts (event) )
  contact_ops      allow    (has_contact (event, "ops") )
```

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 <u>Slack handler dynamic runtime asset</u> to Sensu with sensuctl:

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

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-slack-handler:1.0.3
added asset: sensu/sensu-slack-handler:1.0.3

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-slack-handler"].
```

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

In each handler definition, you will 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` dynamic runtime asset

Before you run the following code to create the handlers with sensuctl, make these changes:

- ⌐ Replace `<alert-ops>`, `<alert-dev>`, and `<alert-all>` with the names of the channels you want to use to receive alerts in your Slack instance.
- ⌐ Replace `<slack_webhook_url>` with your Slack webhook URL.

After you update the code to use your preferred Slack channels and webhook URL, run:

TEXT

```
echo '---
type: Handler
api_version: core/v2
metadata:
  name: slack_ops
spec:
  command: sensu-slack-handler --channel "#<alert-ops>"
  env_vars:
  - SLACK_WEBHOOK_URL=<slack_webhook_url>"
  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=<slack_webhook_url>"
  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=<slack_webhook_url>"
  filters:
  - is_incident
  - not_silenced
  - contact_fallback
  runtime_assets:
  - sensu-slack-handler
  type: pipe' | sensuctl create
```

TEXT

```
echo '{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack_ops"
  },
  "spec": {
```

```json
      "command": "sensu-slack-handler --channel "#<alert-ops>",
      "env_vars": [
        "SLACK_WEBHOOK_URL=<slack_webhook_url>"
      ],
      "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=<slack_webhook_url>"
      ],
      "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=<slack_webhook_url>"
    ],
    "filters": [
      "is_incident",
      "not_silenced",
      "contact_fallback"
    ],
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "type": "pipe"
  }
}' | sensuctl create
```

Just like the event filters, you can save these handlers to a YAML or JSON file to create a handlers configuration file if you're implementing monitoring as code.

Use sensuctl to confirm that the handlers were added:

```
sensuctl handler list
```

The response should list the new `slack_ops`, `slack_dev`, and `slack_fallback` handlers:

Name	Type	Timeout	Filters	Mutator	Execute

Environment Variables Assets

```
_____ _____ _____ _____
_____ _____
_____
_____
 slack_dev     pipe      0  is_incident,not_silenced,contact_dev         RUN:   sensu-slack-handler --channel "#alert-
dev"   SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX     sensu-slack-handler
 slack_fallback  pipe      0  is_incident,not_silenced,contact_fallback       RUN:   sensu-slack-handler --channel
"#alert-all"   SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX     sensu-slack-
handler
```

| slack_ops | pipe | 0 | is_incident,not_silenced,contact_ops | RUN: sensu-slack-handler --channel "#alert-ops" SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX sensu-slack-handler |

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, `slack`:

TEXT

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

TEXT

```
echo '{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "handlers": [
      "slack_ops",
      "slack_dev",
      "slack_fallback"
    ],
    "type": "set"
  }
```

```
}' | sensuctl create
```

Run `sensuctl handler list` again.The updated output should include the `slack` handler set:

```
    Name       Type  Timeout              Filters           Mutator              Execute
  Environment Variables                   Assets
  _____ _____ _____ _____
  _____ _____
  _____
  _____
   slack        set       0                              CALL:  slack_ops,slack_dev,slack_fallback
   slack_dev      pipe      0  is_incident,not_silenced,contact_dev          RUN:   sensu-slack-handler --channel "#alert-
  dev"  SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX   sensu-slack-handler
   slack_fallback  pipe      0  is_incident,not_silenced,contact_fallback       RUN:   sensu-slack-handler --channel
  "#alert-all"  SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX   sensu-slack-handler
   slack_ops      pipe      0  is_incident,not_silenced,contact_ops          RUN:   sensu-slack-handler --channel "#alert-
  ops"  SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX   sensu-slack-handler
```

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 receive a 202 response from the API. Since this event doesn't include a `contacts` label, you should also receive 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 receive 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. The `contacts` labels should be `ops` and `dev`. You'll also need to update the check to use your `slack` handler.

For example, you can update the `check_cpu` check created in [Monitor server resources](#) to include the `ops` and `dev` contacts and the `slack` handler.

Use sensuctl to open the check in a text editor:

```
sensuctl edit check check_cpu
```

Edit the check metadata to add the following labels:

YML

```
---
labels:
  contacts: ops, dev
```

JSON

```
{
  "labels": {
    "contacts": "ops, dev"
  }
}
```

Save and close the updated check definition. A response will confirm the check was updated. For example:

```
Updated /api/core/v2/namespaces/default/checks/check_cpu
```

Next, run this sensuctl command to add the `slack` handler:

```
sensuctl check set-handlers check_cpu slack
```

Again, you will receive an `Updated` confirmation message.

To view the updated resource definition for `check_cpu` and confirm that it includes the `contacts` labels and `slack` handler, run:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The sensuctl response will include the updated `check_cpu` resource definition in the specified format:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  labels:
    contacts: ops, dev
  name: check_cpu
  namespace: default
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers:
  - slack
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
```

```
    proxy_entity_name: ""
    publish: true
    round_robin: false
    runtime_assets:
    - check-cpu-usage
    secrets: null
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "labels": {
      "contacts": "ops, dev"
    },
    "name": "check_cpu",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
```

```
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

Now when the `check_cpu` check generates an incident, Sensu will filter the event according to the `contact_ops` and `contact_dev` event filters and send alerts to #alert-ops and #alert-dev accordingly.
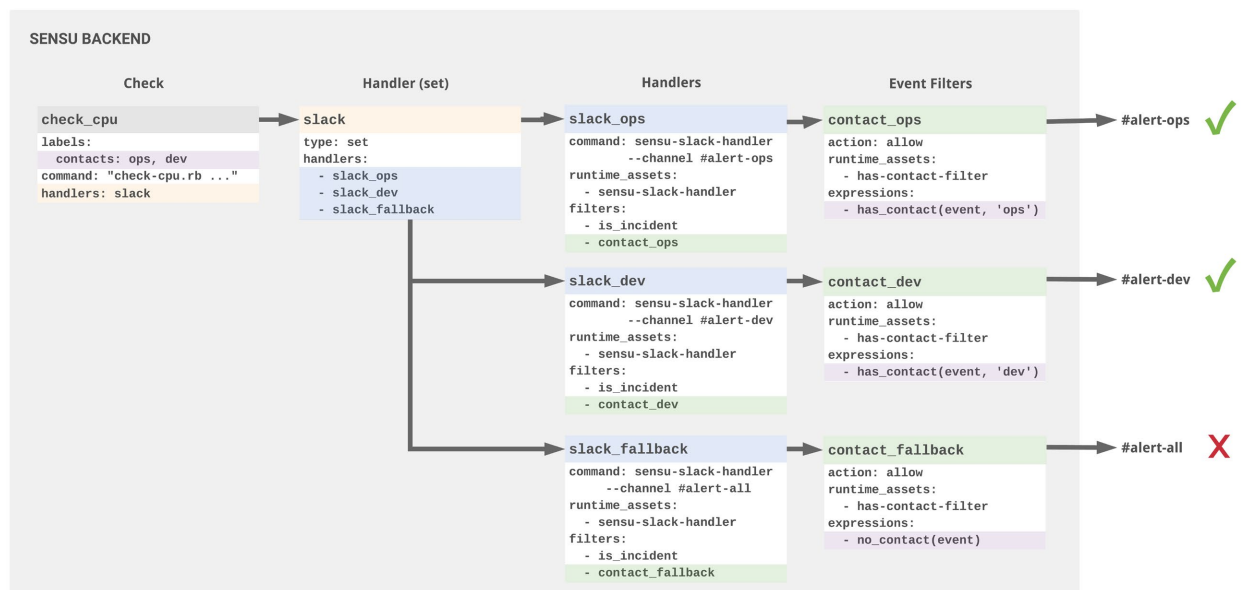
Entities

You can also specify contacts using an entity label. For more information about managing entity labels, read 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.
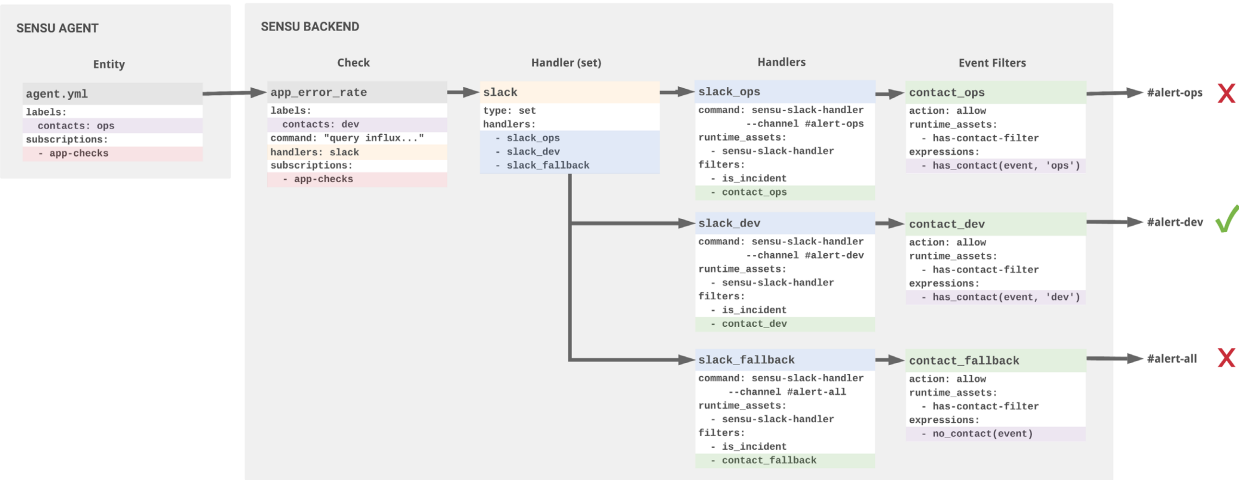
Next steps

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 Reduce alert fatigue.

Sensu query expressions reference

Sensu query expressions (SQEs) are JavaScript-based expressions that provide additional functionality for using Sensu, like nested parameters and custom functions.

SQEs are defined in _event filters_, so they act in the context of determining whether a given event should be passed to the handler. SQEs always receive a single event and some information about that event, like `event.timestamp` or `event.check.interval` .

SQEs always return either `true` or `false` . They are evaluated by the _Otto JavaScript VM_ as JavaScript programs.

Syntax quick reference

operator	description
`===`	Identity
`!==`	Nonidentity
`==`	Equality
`!=`	Inequality
`&&`	Logical AND
`||`	Logical OR
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to

Specification

SQEs are valid ECMAScript 5 (JavaScript) expressions that return either `true` or `false`. Other values are not allowed. If an SQE returns a value besides `true` or `false`, an error is recorded in the Sensu backend log and the filter evaluates to `false`.

Custom functions

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
```

sensu.CheckDependencies

The `sensu.CheckDependencies` SQE takes zero or more checks as arguments. It returns `true` if all the specified checks are passing or `false` if any of the specified checks are failing.

If you do not specify any checks, the `sensu.CheckDependencies` SQE always returns `true`.

You can refer to the checks as strings. In this example, if all checks named `database` or `disk` are passing, the following SQE returns `true`:

```
sensu.CheckDependencies("database", "disk")
```

If you pass the check names as strings, Sensu assumes that the entities are the same as those in the events being filtered.

You can also refer to the checks in objects that include both the entity and check name. For example:

```
sensu.CheckDependencies({entity: "server01", check: "disk"}, {entity: "server01", check: "database"})
```

In both cases, if no event matches the specified entities and checks, Sensu will raise an error.

Examples

Evaluate an event attribute

This SQE returns `true` if the event's entity contains a custom attribute named `namespace` that is equal to `production`:

```
event.entity.namespace == 'production'
```

Evaluate an array

To evaluate an attribute that contains an array of elements, use the `.indexOf` method. For example, this expression returns `true` if an entity includes the subscription `system`:

```
entity.subscriptions.indexOf('system') >= 0
```

Evaluate the day of the week

This expression returns `true` if the event occurred on a weekday:

```
weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5
```

Evaluate office hours

This expression returns `true` if the event occurred between 9 AM and 5 PM UTC:

```
hour(event.timestamp) >= 9 && hour(event.timestamp) <= 17
```

Evaluate labels and annotations

Although you can use annotations to create SQEs, we recommend using labels because labels provide identifying information.

This expression returns `true` if the event's entity includes the label `webserver`:

```
!!event.entity.labels.webserver
```

Likewise, this expression returns `true` if the event's entity includes the annotation `www.company.com`:

```
!!event.entity.annotations['www.company.com']
```

Transform your observation data

Sensu Observability Pipeline
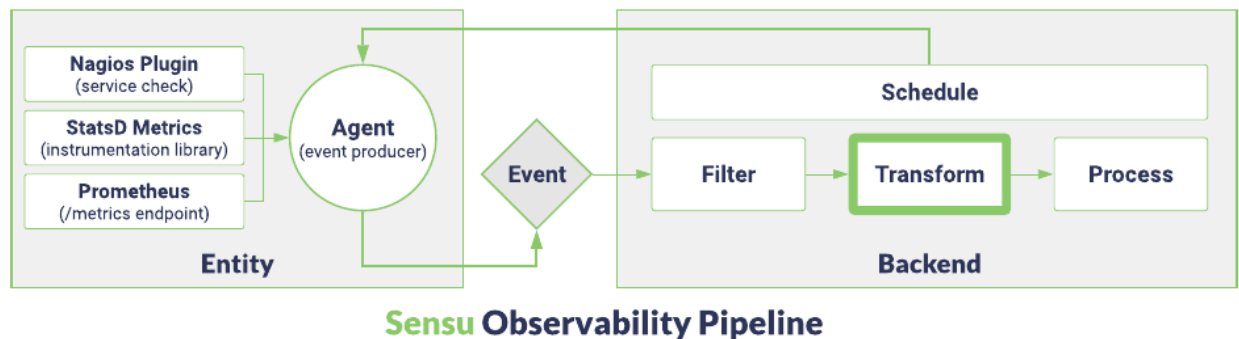
In the transform stage, Sensu executes <u>mutators</u>.

The transform stage of the Sensu observability pipeline executes any <u>mutators</u> you have specified in your <u>handler</u> configuration to transform your observability data so other technologies can consume it. For example, if you're sending metrics to Graphite using a TCP handler, Graphite expects data that follows the Graphite plaintext protocol. You can add Sensu's <u>built-in only_check_output mutator</u> to transform the data into the format Graphite can accept.

Here's how transform stage of the pipeline works: first, the Sensu backend receives an event and executes the <u>filter</u> stage of the observability pipeline. If the event data meets the conditions, triggers, or thresholds you specified in your event filters, Sensu checks the handler for a mutator. If the handler includes a mutator, the Sensu backend executes the mutator.

- ￢ If the mutator executes successfully (that is, returns an exit status code of `0`), Sensu applies the mutator to transform the event data, returns the transformed event data to the handler, and executes the handler.

- ￢ If the mutator fails to execute (that is, returns a non-zero exit status code or fails to complete within its configured timeout), Sensu logs an error and does not execute the handler.

This example mutator resource definition uses the <u>Sensu Check Status Metric Mutator</u> dynamic runtime asset:

YML

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

```yaml
metadata:
  name: sensu-check-status-metric-mutator
  namespace: default
spec:
  command: sensu-check-status-metric-mutator
  runtime_assets:
  - nixwiz/sensu-check-status-metric-mutator
```

JSON

```json
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-check-status-metric-mutator",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-check-status-metric-mutator",
    "runtime_assets": [
      "nixwiz/sensu-check-status-metric-mutator"
    ]
  }
}
```

Use Bonsai, the Sensu asset hub, to discover, download, and share Sensu mutator dynamic runtime assets.Read Use assets to install plugins to get started.

Mutators reference

Sensu executes mutators during the **transform** stage of the observability pipeline.

Handlers can specify a mutator to execute and transform observability event data before any handlers are applied. When the Sensu backend processes an event, it checks the handler for the presence of a mutator and executes that mutator before executing the handler.

Mutators produce an exit status code to indicate state. A code of `0` indicates OK status. If the mutator executes successfully (returns an exit status code of `0`), the modified event data return to the handler and the handler is executed.

Exit codes other than `0` indicate failure. If the mutator fails to execute (returns a non-zero exit status code or fails to complete within its configured timeout), an error is logged and the handler will not execute.

Mutators accept input/data via `STDIN` and can parse JSON event data. They output JSON data (modified event data) to `STDOUT` or `STDERR`.

Mutator examples

This example shows a mutator resource definition with the minimum required attributes:

YML

```
type: Mutator
api_version: core/v2
metadata:
  name: mutator_minimum
  namespace: default
spec:
  command: example_mutator.go
```

JSON

```
{
  "type": "Mutator",
```

```json
    "api_version": "core/v2",
    "metadata": {
      "name": "mutator_minimum",
      "namespace": "default"
    },
    "spec": {
      "command": "example_mutator.go"
    }
  }
```

The following 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": []
    }
  }
```

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

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	**YML**

```yml
type: Mutator
```

JSON

```json
{
  "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	**YML**

```yml
api_version: core/v2
```

JSON

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

metadata

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

YML

```yaml
metadata:
  name: example-mutator
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    slack-channel: "#monitoring"
```

JSON

```json
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
```

```
        "region": "us-west-1"
      },
      "annotations": {
        "slack-channel": "#monitoring"
      }
    }
  }
```

spec	
description	Top-level map that includes the mutator spec attributes.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs
example	**YML**

```yaml
spec:
  command: example_mutator.go
  timeout: 0
  env_vars: []
  runtime_assets: []
```

JSON

```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

YML

```yml
name: example-mutator
```

JSON

```json
{
  "name": "example-mutator"
}
```

namespace	
description	Sensu RBAC namespace that the mutator belongs to.
required	false
type	String
default	`default`

example

YML

```yml
namespace: production
```

JSON

```json
{
```

```
      "namespace": "production"
  }
```

created_by

description	Username of the Sensu user who created the mutator or last updated the mutator. Sensu automatically populates the `created_by` field when the mutator is created or updated.
required	false
type	String
example	**YML**

```
  created_by: admin
```

JSON

```
  {
    "created_by": "admin"
  }
```

labels

description	Custom attributes to include with event data that you can use for response and web UI view filtering.
	If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data.
	Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.

required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example

YML

```yaml
labels:
  environment: development
  region: us-west-2
```

JSON

```json
{
  "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 web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example

YML

```yaml
annotations:
```

```
  managed-by: ops
  playbook: www.example.url
```

JSON

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

Spec attributes

command	
description	Mutator command to be executed by the Sensu backend.
required	true
type	String

example	**YML**
	```command: /etc/sensu/plugins/mutated.go```
	JSON
	```{ "command": "/etc/sensu/plugins/mutated.go" }```

timeout	

description	Mutator execution duration timeout (hard stop). In seconds.
required	false
type	integer

example

YML

```
timeout: 30
```

JSON

```
{
  "timeout": 30
}
```

env_vars

description	Array of environment variables to use with command execution.
required	false
type	Array

example

YML

```
env_vars:
- RUBY_VERSION=2.5.0
```

JSON

```
{
  "env_vars": [
    "RUBY_VERSION=2.5.0"
  ]
}
```

runtime_assets

description	Array of <u>Sensu dynamic runtime assets</u> (by their names) required at runtime for execution of the `command`.
required	false
type	Array
example	**YML**

```yml
runtime_assets:
- ruby-2.5.0
```

JSON

```json
{
  "runtime_assets": [
    "ruby-2.5.0"
  ]
}
```

secrets

description	Array of the name/secret pairs to use with command execution.
required	false
type	Array
example	**YML**

```yml
secrets:
- name: ANSIBLE_HOST
  secret: sensu-ansible-host
- name: ANSIBLE_TOKEN
  secret: sensu-ansible-token
```

JSON

```json
{
  "secrets": [
    {
      "name": "ANSIBLE_HOST",
      "secret": "sensu-ansible-host"
    },
    {
      "name": "ANSIBLE_TOKEN",
      "secret": "sensu-ansible-token"
    }
  ]
}
```

`secrets` *attributes*

name	
description	Name of the secret defined in the executable command. Becomes the environment variable presented to the mutator. Read Use secrets management in Sensu for more information.
required	true
type	String

example	**YML**
	
	``yaml
	name: ANSIBLE_HOST
	``
	
	JSON
	
	``json
	{
	  "name": "ANSIBLE_HOST"
	}
	``

secret	
description	Name of the Sensu secret resource that defines how to retrieve the secret.
required	true
type	String

example	**YML**
	
	``yml
	secret: sensu-ansible-host
	``
	
	JSON
	
	``json
	{
	  "secret": "sensu-ansible-host"
	}
	``

Mutator that uses secrets management

Learn more about secrets management for your Sensu configuration in the secrets and secrets providers references.

YML

```yml
---
type: Mutator
api_version: core/v2
metadata:
  name: ansible-tower
  namespace: ops
spec:
  command: sensu-ansible-mutator -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN
  secrets:
  - name: ANSIBLE_HOST
    secret: sensu-ansible-host
  - name: ANSIBLE_TOKEN
    secret: sensu-ansible-token
```

JSON

```json
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "ansible-tower",
    "namespace": "ops"
  },
  "spec": {
    "command": "sensu-ansible-mutator -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN",
    "secrets": [
      {
        "name": "ANSIBLE_HOST",
        "secret": "sensu-ansible-host"
      },
      {
        "name": "ANSIBLE_TOKEN",
        "secret": "sensu-ansible-token"
      }
    ]
  }
}
```

Process your observation data

Sensu Observability Pipeline
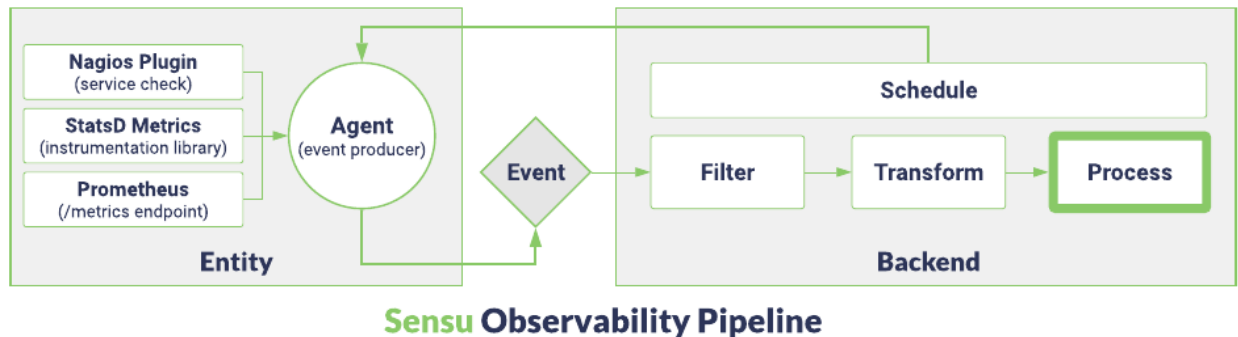
In the process stage, Sensu executes <u>handlers</u>.

In the process stage of Sensu's observability pipeline, the Sensu backend executes <u>handlers</u> to take action on your observation data. Your handler configuration determines what happens to the events that comes through your pipeline. For example, your handler might route incidents to a specific Slack channel or PagerDuty notification workflow, or send metrics to InfluxDB or Prometheus.

Sensu also checks your handlers for the event filters and mutators to apply in the <u>filter</u> and <u>transform</u> stages.

A few different types of handlers are available in Sensu. The most common are <u>pipe handlers</u>, which work similarly to <u>checks</u> and enable Sensu to interact with almost any computer program via <u>standard streams</u>.

Here's an example resource definition for a pipe handler — read <u>Send Slack alerts with handlers</u> to learn how to configure your own version of this handler:

YML

```yml
---
type: Handler
api_version: core/v2
metadata:
  created_by: admin
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
```

```yaml
    env_vars:
    - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXX
    filters: null
    handlers: null
    runtime_assets:
    - sensu-slack-handler
    secrets: null
    timeout: 0
    type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXX"
    ],
    "filters": null,
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

You can also use TCP/UDP handlers to send your observation data to remote sockets and handler sets to streamline groups of actions to execute for certain types of events.

Discover, download, and share Sensu handler dynamic runtime assets in Bonsai, the Sensu asset hub

Read Use assets to install plugins to get started.

Handlers reference

Sensu executes handlers during the **process** stage of the observability pipeline.

Handlers are actions the Sensu backend executes on events. Several types of handlers are available. The most common are `pipe` handlers, which work similarly to checks and enable Sensu to interact with almost any computer program via standard streams.

- ⌐ **Pipe handlers** send observation data (events) into arbitrary commands via `STDIN`
- ⌐ **TCP/UDP handlers** send observation data (events) to a remote socket
- ⌐ **Handler sets** group event handlers and streamline groups of actions to execute for certain types of events (also called "set handlers")

The handler stack concept describes a group of handlers or a handler set that escalates events through a series of different handlers.

Discover, download, and share Sensu handler dynamic runtime assets using Bonsai, the Sensu asset hub. Read Use dynamic runtime assets to install plugins to get started.

Pipe handlers

Pipe handlers are external commands that can consume event data via STDIN.

Pipe handler example

This example shows a pipe handler resource definition with the minimum required attributes:

YML

```yaml
---
type: Handler
api_version: core/v2
metadata:
  name: pipe_handler_minimum
  namespace: default
```

```yaml
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"
  }
}
```

Pipe handler command

Pipe handler definitions include a `command` attribute, which is a command for the Sensu backend to execute.

Pipe handler command arguments

Pipe handler `command` attributes may include command line arguments for controlling the behavior of the `command` executable.

TCP/UDP handlers

TCP and UDP handlers enable Sensu to forward event data to arbitrary TCP or UDP sockets for external services to consume.

TCP/UDP handler example

This handler will send event data to a TCP socket (10.0.1.99:4444) and timeout if an acknowledgement (`ACK`) is not received within 30 seconds:

YML

```yml
---
type: Handler
api_version: core/v2
metadata:
  name: tcp_handler
  namespace: default
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: tcp
  timeout: 30
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "tcp_handler",
    "namespace": "default"
  },
  "spec": {
    "type": "tcp",
    "timeout": 30,
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Change the `type` from `tcp` to `udp` to configure a UDP handler:

YML

```yaml
---
type: Handler
api_version: core/v2
metadata:
  name: udp_handler
  namespace: default
spec:
  socket:
    host: 10.0.1.99
    port: 4444
  type: udp
  timeout: 30
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "udp_handler",
    "namespace": "default"
  },
  "spec": {
    "type": "udp",
    "timeout": 30,
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Handler sets

Handler set definitions allow you to use a single named handler set to refer to groups of handlers. The handler set becomes a collection of individual actions to take (via each included handler) on event data.

For example, suppose you have already created these two handlers:

- `elasticsearch` to send all observation data to Elasticsearch.

- `opsgenie` to send non-OK status alerts to your OpsGenie notification channel.

You can list both of these handlers in a handler set to automate and streamline your workflow, specifying `type: set`:

YML

```yaml
type: Handler
api_version: core/v2
metadata:
  name: send_events_notify_operator
  namespace: default
spec:
  handlers:
  - elasticsearch
  - opsgenie
  type: set
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "send_events_notify_operator",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "elasticsearch",
      "opsgenie"
    ]
  }
}
```

Now you can route observation data to Elasticsearch and alerts to OpsGenie with a single handler definition, the `send_events_notify_operator` handler set.

> **NOTE**: *Attributes defined in handler sets do not apply to the handlers they include. For example, `filters` and `mutator` attributes defined in a handler set will have no effect on handlers. Define these attributes in individual handlers instead.*

Handler stacks

The handler stack concept refers to a group of handlers or a handler set that escalates events through a series of different handlers. For example, suppose you want a handler stack with three levels of escalation:

- Level 1: On the first occurrence, attempt remediation.

- Level 2: On the fifth occurrence, send an alert to Slack.

- Level 3: On the tenth occurrence, send an alert to PagerDuty. Continue to send this alert on every tenth occurrence thereafter until the incident is resolved.

A handler stack for this scenario requires three handlers to take the desired actions based on three corresponding event filters that control the escalation levels:

- Level 1 requires an event filter with the built-in `is_incident` filter plus an occurrence-based filter that uses an expression like `event.check.occurrences ==1` and a corresponding remediation handler.

- Level 2 requires an event filter with `is_incident` plus an occurrence-based filter that uses an expression like `event.check.occurrences == 5` and a corresponding Slack handler.

- Level 3 requires an event filter with `is_incident` plus an occurrence-based filter that uses an expression like `event.check.occurrences % 10 == 0` to match event data with an occurrences value that is evenly divisible by 10 via a modulo operator calculation and a corresponding PagerDuty handler.

With these event filters and handlers configured, you can create a handler set that includes the three handlers in your stack. You can also list the three handlers in the handlers array in your check definition instead.

> **PRO TIP**: *This scenario relies on six different resources, three event filters and three handlers, to describe the handler stack concept, but you can use Sensu dynamic runtime assets and integrations to achieve the same escalating alert levels in other ways.*

For example, you can use the `is_incident` *event filter in conjunction with the [Sensu Go Fatigue Check Filter](#) asset to control event escalation.Sensu's [Ansible](#), [Rundeck](#), and [Saltstack](#) auto-remediation integrations and the [Sensu Remediation Handler](#) asset also include built-in occurrence- and severity-based event filtering.*

Keepalive event handlers

Sensu [keepalives](#) are the heartbeat mechanism used to ensure that all registered [Sensu agents](#) are operational and can reach the [Sensu backend](#).You can connect keepalive events to your monitoring workflows using a keepalive handler.Sensu looks for an event handler named `keepalive` and automatically uses it to process keepalive events.

Suppose you want to receive Slack notifications for keepalive alerts, and you already have a [Slack handler set up to process events](#).To process keepalive events using the Slack pipeline, create a handler set named `keepalive` and add the `slack` handler to the `handlers` array.The resulting `keepalive` handler set configuration will look like this example:

YML

```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"
      ]
    }
  }
```

You can also use the `keepalive-handlers` flag to send keepalive events to any handler you have configured.If you do not specify a keepalive handler with the `keepalive-handlers` flag, the Sensu backend will use the default `keepalive` handler and create an event in sensuctl and the Sensu web UI.

Handler specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Handlers should always be type `Handler` .
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	String
| example | **YML**<br><br>```yaml
type: Handler
```<br><br>**JSON**<br><br>```json
{
  "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

YML
```yaml
api_version: core/v2
```

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

metadata

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

YML
```yaml
metadata:
  name: handler-slack
  namespace: default
```

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

JSON

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

spec	
description	Top-level map that includes the handler spec attributes.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs
	YML
example	

```yaml
  spec:
    type: tcp
    socket:
      host: 10.0.1.99
      port: 4444
    metadata:
```

```yaml
    name: tcp_handler
    namespace: default
```

```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	YML

```yaml
    name: handler-slack
```

JSON

```json
{
```

```
      "name": "handler-slack"
    }
```

namespace

description	Sensu RBAC namespace that the handler belongs to.
required	false
type	String
default	`default`

YML

example	

```
  namespace: production
```

JSON

```
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the handler or last updated the handler. Sensu automatically populates the `created_by` field when the handler is created or updated.
required	false
type	String

YML

example	

```
  created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation data in events that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null` YML

example	```yaml
	labels:
	  environment: development
	  region: us-west-2
	``
	
	JSON
	
	``json
	{
	  "labels": {
	    "environment": "development",
	    "region": "us-west-2"
	  }
	``

```
    }
```

annotations

description	Non-identifying metadata to include with observation data in events that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

YML

example

```yml
annotations:
  managed-by: ops
  playbook: www.example.url
```

JSON

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

Spec attributes

type

description	Handler type.
required	true
type	String
allowed values	`pipe` , `tcp` , `udp` , and `set`

YML

```yml
type: pipe
```

JSON

```json
{
  "type": "pipe"
}
```

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

YML

```yml
filters:
- occurrences
- production
```

JSON

```json
{
  "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

YML

```
  mutator: only_check_output
```

JSON

```
  {
    "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

YML

```
  timeout: 30
```

JSON

```json
{
  "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 (that is, handlers configured with `"type": "pipe"`).
required	true (if `type` equals `pipe`)
type	String

example

YML

```yml
command: /etc/sensu/plugins/pagerduty.go
```

JSON

```json
{
  "command": "/etc/sensu/plugins/pagerduty.go"
}
```

env_vars

description	Array of environment variables to use with command execution. **NOTE**: The `env_vars` attribute is only supported for pipe handlers

(that is, handlers configured with `"type": "pipe"` *).*

required	false
type	Array

YML

example	

```yml
env_vars:
- API_KEY=0428d6b8nb51an4d95nbe28nf90865a66af5
```

JSON

```json
{
  "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 (that is, handlers configured with `"type": "tcp"` or `"type": "udp"`).

required	true (if `type` equals `tcp` or `udp`)
type	Hash

YML

example	

```yml
socket: {}
```

JSON

```
{
  "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 (that is, handlers configured with `"type": "set"`).

required	true (if `type` equals `set`)
type	Array

YML

```yaml
handlers:
- pagerduty
- email
- ec2
```

JSON

```json
{
  "handlers": [
    "pagerduty",
    "email",
    "ec2"
  ]
}
```

runtime_assets

description	Array of Sensu dynamic runtime assets (by names) required at runtime to execute the `command`
required	false
type	Array

example

YML

```yml
runtime_assets:
- ruby-2.5.0
```

JSON

```json
{
  "runtime_assets": [
    "ruby-2.5.0"
  ]
}
```

secrets

description	Array of the name/secret pairs to use with command execution.
required	false
type	Array

example

YML

```yml
secrets:
- name: ANSIBLE_HOST
  secret: sensu-ansible-host
- name: ANSIBLE_TOKEN
  secret: sensu-ansible-token
```

JSON

```json
{
  "secrets": [
    {
      "name": "ANSIBLE_HOST",
      "secret": "sensu-ansible-host"
    },
    {
      "name": "ANSIBLE_TOKEN",
      "secret": "sensu-ansible-token"
    }
  ]
}
```

`socket` *attributes*

host	
description	Socket host address (IP or hostname) to connect to.
required	true
type	String
example	**YML**

```yml
host: 8.8.8.8
```

JSON

```json
{
  "host": "8.8.8.8"
}
```

port	

description	Socket port to connect to.
required	true
type	Integer

YML

```yml
port: 4242
```

JSON

```json
{
  "port": 4242
}
```

`secrets` *attributes*

name	
description	Name of the secret defined in the executable command. Becomes the environment variable presented to the handler. Read Use secrets management in Sensu for more information.
required	true
type	String

YML

```yml
name: ANSIBLE_HOST
```

JSON

```json
{
  "name": "ANSIBLE_HOST"
}
```

secret	
description	Name of the Sensu secret resource that defines how to retrieve the <u>secret</u>.
required	true
type	String
example	**YML**

```
secret: sensu-ansible-host
```

JSON

```
{
  "secret": "sensu-ansible-host"
}
``` |

Send Slack alerts

This handler will send alerts to a channel named `monitoring` with the configured webhook URL, using the `handler-slack` executable command.

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",
    "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 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

```yaml
---
type: Handler
api_version: core/v2
metadata:
  name: registration
  namespace: default
spec:
  handlers:
  - servicenow-cmdb
  type: set
```

JSON

```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 (handler set)

The following example creates a handler set, `notify_all_the_things` , that will execute three handlers: `slack` , `tcp_handler` , and `udp_handler` .

YML

```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

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "notify_all_the_things",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack",
      "tcp_handler",
      "udp_handler"
    ]
  }
}
```

Use secrets management in a handler

Learn more about <u>secrets management</u> for your Sensu configuration in the <u>secrets</u> and <u>secrets providers</u> references.

YML

```yml
---
type: Handler
api_version: core/v2
metadata:
  name: ansible-tower
  namespace: ops
spec:
  type: pipe
  command: sensu-ansible-handler -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN
  secrets:
  - name: ANSIBLE_HOST
    secret: sensu-ansible-host
  - name: ANSIBLE_TOKEN
    secret: sensu-ansible-token
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "ansible-tower",
    "namespace": "ops"
  },
  "spec": {
    "type": "pipe",
    "command": "sensu-ansible-handler -h $ANSIBLE_HOST -t $ANSIBLE_TOKEN",
    "secrets": [
      {
        "name": "ANSIBLE_HOST",
        "secret": "sensu-ansible-host"
      },
      {
        "name": "ANSIBLE_TOKEN",
```

```
            "secret": "sensu-ansible-token"
        }
    ]
    }
}
```

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 dynamic runtime 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 dynamic runtime asset

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

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

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

The response will confirm that the asset was added:

```
no version specified, using latest: 0.6.0
fetching bonsai asset: sensu/sensu-email-handler:0.6.0
added asset: sensu/sensu-email-handler:0.6.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
```

```
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["email-handler"].
```

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

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

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

```
sensuctl asset list
```

The list should include the `email-handler` dynamic runtime asset. For a detailed list of everything
related to the asset that Sensu added automatically, run:

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

The dynamic runtime asset includes the `sensu-email-handler` command, which you will use when
you <u>create the email handler definition</u> later in this guide.

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until
> they are needed for command execution. Read <u>the asset reference</u> for more information about
> dynamic runtime asset builds.

Add an event filter

Event filters allow you to fine-tune how your events are handled and <u>reduce alert fatigue</u>. In this guide,
your event filter will send notifications only when your event's state changes (for example, for any
change between `0` OK, `1` warning, and `2` critical).

Here's an overview of how the `state_change_only` filter will work:

- ¬ If your event status changes from `0` to `1`, you will receive **one** email notification for the change to warning status.

- ¬ If your event status stays at `1` for the next hour, you **will not** receive repeated email notifications during that hour.

- ¬ If your event status changes to `2` after 1 hour at `1`, you will receive **one** email notification for the change from warning to critical status.

- ¬ If your event status fluctuates between `0`, `1`, and `2` for the next hour, you will receive **one** email notification **each time** the status changes.

To create the event filter, run:

TEXT

```
cat << EOF | sensuctl create
---
type: EventFilter
api_version: core/v2
metadata:
  annotations: null
  labels: null
  name: state_change_only
  namespace: default
spec:
  action: allow
  expressions:
  - event.check.occurrences == 1
  runtime_assets: []
EOF
```

TEXT

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

- `<sender@example.com>` : Replace with the email address you want to use to send email
  alerts.

- `<recipient@example.com>` : Replace with the email address you want to receive email alerts.

- `<smtp_server@example.com>` : Replace with the hostname of your SMTP server.

- `<username>` : Replace with your SMTP username, typically your email address.

- `<password>` : Replace with your SMTP password, typically the same as your email password.

> **NOTE**: To use Gmail or G Suite as your SMTP server, follow Google's instructions to _send email_
> _via SMTP_. If you have enabled 2-step verification on your Google account, use an _app password_
> instead of your login password. If you have not enabled 2-step verification, you may need to adjust
> your _app access settings_ to follow the example in this guide.

With your updates, create the handler using sensuctl:

TEXT

```
cat << EOF | sensuctl create
---
api_version: core/v2
```

```
type: Handler
metadata:
  namespace: default
  name: email
spec:
  type: pipe
  command: sensu-email-handler -f <sender@example.com> -t <recipient@example.com> -s
<smtp_server@example.com> -u username -p password
  timeout: 10
  filters:
  - is_incident
  - not_silenced
  - state_change_only
  runtime_assets:
  - email-handler
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "api_version": "core/v2",
  "type": "Handler",
  "metadata": {
    "namespace": "default",
    "name": "email"
  },
  "spec": {
    "type": "pipe",
    "command": "sensu-email-handler -f <sender@example.com> -t
<recipient@example.com> -s <smtp_server@example.com> -u username -p password",
    "timeout": 10,
    "filters": [
      "is_incident",
      "not_silenced",
      "state_change_only"
    ],
    "runtime_assets": [
      "email-handler"
    ]
  }
}
```

```
EOF
```

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 dynamic runtime asset makes it possible to add a template that provides context for your email notifications. The email template functionality uses tokens to populate the values provided by the event, and you can use HTML to format the email.

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 Sensu API access token credential for the Sensu API:

```
eval $(sensuctl env)
```

Verify that the `SENSU_ACCESS_TOKEN` environment variable is set by echoing its value:

```
echo $SENSU_ACCESS_TOKEN
efPxbRciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NzkwMzY5NjQsImp0aSI6ImJiMmY0ODY4ZTJ
hZWEyMDhhMTExOTllMGZkNzkzMDc0Iiwic3ViIjoiYWRtaW4iLCJncm91cHMiOlsiY2x1c3Rlci1hZG1pbnM
iLCJzeXN0ZW06dXNlcnMiXSwicHJvdmlkZXIiOnsicHJvdmlkZXJfaWQiOiJiYXNpYyIsInByb3ZpZGVyX3R
5cGUiOiIiLCJ1c2VyX2lkIjoiYWRtaW4ifX0.6XmuvblCN743R2maF4yErS3K3sOVczsCBsjib9TenUU
```

With the environment variables set, you can use the Sensu API to create your ad hoc observability 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 view the results!

To generate a status change event, use the update event endpoint to create a `1` (warning) event.Run:

```
curl -sS -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server01",
      "namespace": "default"
    }
  },
  "check": {
    "metadata": {
```

```
        "name": "server-health"
      },
      "output": "This is a warning.",
      "status": 1,
      "interval": 60,
      "handlers": ["email"]
    }
  }' \
  http://localhost:8080/api/core/v2/namespaces/default/events/server01/server-health
```

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

Check your email — you should receive a message from Sensu!

Create another event with status set to `0`. Run:

```
curl -sS -X PUT \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server01",
      "namespace": "default"
    }
  },
  "check": {
    "metadata": {
      "name": "server-health"
    },
    "output": "Everything is OK.",
    "status": 0,
    "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.

- Learn how to use the event filter and handler resources you created to start developing a monitoring as code repository.

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

- Check out the Reduce alert fatigue guide.

Send PagerDuty alerts with Sensu

Sensu checks are commands the Sensu agent executes that generate observability data in a status or metric event. Sensu handlers define the actions the Sensu backend executes on the events. Follow this guide to create a check that looks for a specific file and a handler that sends an alert to PagerDuty if the file is not found.

One quick note before you begin: you'll need your PagerDuty API integration key to set up the handler in this guide.

Install and configure Sensu Go

Follow the RHEL/CentOS install instructions to install and configure the Sensu backend, the Sensu agent, and sensuctl.

Find your entity name:

```
sensuctl entity list
```

The `ID` in the response is the name of your entity.

Replace `<entity_name>` with the name of your entity in the sensuctl command below. Then run the command to add the `system` subscription to your entity:

```
sensuctl entity update <entity_name>
```

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

Confirm both Sensu services are running:

```
systemctl status sensu-backend && systemctl status sensu-agent
```

Add the file_exists check

Before Sensu can send alerts to your PagerDuty account, you'll need a check to generate events. In this guide, you will use a `file_exists` check, which is included in Sensu's Nagios Foundation dynamic runtime asset.

Before you can add the check, you need to add the Nagios Foundation asset to your Sensu configuration:

```
sensuctl asset add ncr-devops-platform/nagiosfoundation
```

To confirm that the asset was added to your Sensu backend, run:

```
sensuctl asset info ncr-devops-platform/nagiosfoundation
```

The list should include `=== ncr-devops-platform/nagiosfoundation` followed by a list of available builds for the asset.

Now that you've added the Nagios Foundation dynamic runtime asset, you can add its `file_exists` check to your Sensu backend. Use sensuctl to add the check:

```
sensuctl check create file_exists \
--command "check_file_exists --pattern /tmp/my-file.txt" \
--subscriptions system \
--handlers pagerduty \
--interval 10 \
--runtime-assets ncr-devops-platform/nagiosfoundation
```

To confirm that the check was added to your Sensu backend and view the check definition in YAML or JSON format, run:

SHELL

```
sensuctl check info file_exists --format yaml
```

SHELL

```
sensuctl check info file_exists --format wrapped-json
```

The response will list the complete check resource definition:

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: file_exists
  namespace: default
spec:
  check_hooks: null
  command: check_file_exists --pattern /tmp/my-file.txt
  env_vars: null
  handlers:
  - pagerduty
  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:
  - ncr-devops-platform/nagiosfoundation
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "file_exists",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check_file_exists --pattern /tmp/my-file.txt",
    "env_vars": null,
    "handlers": [
      "pagerduty"
    ],
    "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": [
      "ncr-devops-platform/nagiosfoundation"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
    "ttl": 0
  }
}
```

The check command includes the path for the file that the check will look for on your system, `/tmp/my-file.txt`. For this guide, you'll add `/tmp/my-file.txt` as a temporary file:

```
touch /tmp/my-file.txt
```

In production, you could replace `/tmp/my-file.txt` with the file path or globbing pattern for any existing file or files on your system.

Well done! In the next step, you'll configure your workflow for sending Sensu alerts to your PagerDuty account.

Add the PagerDuty handler

The Sensu PagerDuty Handler dynamic runtime asset includes the scripts you will need to send events to PagerDuty.

To add the PagerDuty handler asset, run:

```
sensuctl asset add sensu/sensu-pagerduty-handler
```

Once again, it's a good idea to confirm that the asset was added to your Sensu backend. Run:

```
sensuctl asset info sensu/sensu-pagerduty-handler
```

The response will list the available builds for the PagerDuty handler dynamic runtime asset.

Now that you've added the Sensu PagerDuty Handler dynamic runtime asset, you can create a handler that uses the asset to send non-OK events to PagerDuty. This requires you to update the handler command by adding your PagerDuty API key.

In the following command, replace `<pagerduty_key>` with your PagerDuty API integration key. Then run the updated command:

```
sensuctl handler create pagerduty \
```

```
  --type pipe \
  --filters is_incident \
  --runtime-assets sensu/sensu-pagerduty-handler \
  --command "sensu-pagerduty-handler -t <pagerduty_key>"
```

> **NOTE**: For checks whose handlers use the Sensu PagerDuty Handler dynamic runtime asset (like the one you've created in this guide), you can use an alternative method for _authenticating and routing alerts based on PagerDuty teams_.
>
> To use this option, list the teams' PagerDuty API integration keys in the handler definition as environment variables or secrets or in the `/etc/default/sensu-backend` configuration file as environment variables. Then, add check or agent annotations to specify which PagerDuty teams should receive alerts based on check events. Sensu will look up the key in the handler definition or backend configuration file that corresponds to the team name in the check annotation to authenticate and send alerts.

Make sure that your handler was added by retrieving the complete handler definition in YAML or JSON format:

SHELL

```
sensuctl handler info pagerduty --format yaml
```

SHELL

```
sensuctl handler info pagerduty --format wrapped-json
```

The response will list the complete handler resource definition:

YAML

```
---
type: Handler
api_version: core/v2
metadata:
  created_by: admin
  name: pagerduty
  namespace: default
spec:
```

```yaml
    command: sensu-pagerduty-handler -t <pagerduty_key>
    env_vars: null
    filters:
    - is_incident
    handlers: null
    runtime_assets:
    - sensu/sensu-pagerduty-handler
    secrets: null
    timeout: 0
    type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "pagerduty",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-pagerduty-handler -t <pagerduty_key>",
    "env_vars": null,
    "filters": [
      "is_incident"
    ],
    "handlers": null,
    "runtime_assets": [
      "sensu/sensu-pagerduty-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

Trigger an event

With your `file_exists` check and PagerDuty handler workflow in place, you can remove a file to cause Sensu to send a non-OK event.

Remove the file `/tmp/my-file.txt`:

```
rm /tmp/my-file.txt
```

This will make sure the file is **not** there for Sensu to find the next time the `file_exists` check runs. After about 10 seconds, Sensu will detect that `my-file.txt` is missing and reflect that in an event.

To view the event with sensuctl, run:

```
sensuctl event list
```

The response should show that the file removal resulted in a CRITICAL (2) event:

```
    Entity      Check                        Output                              Status  Silenced      Timestamp
  UUID
 ────────────── ──────────────                                                                                                                                
                                                                                                                                                              ─────────
 ─────────── ──────────────────────────────────────── ──────────────────────────────────────────
  host01       file_exists   CheckFileExists CRITICAL - 0 files matched pattern /tmp/my-file.txt      2   false     2021-03-15
 19:28:21 +0000 UTC   1b4266ae-7200-4728-a0n4-2f50f7a56613
```

Open the Sensu web UI to view the events the `file_exists` check is generating. Visit http://127.0.0.1:3000, and log in as the admin user (created during initialization when you installed the Sensu backend). The failing check's events will be listed on the Events page.

Observe the alert in PagerDuty

After Sensu detects the non-OK event, the handler you set up will send the alert to PagerDuty. Log in to your PagerDuty account to view an event similar to this one:

	Status	Urgency ▼	Title	Created ⇕	Service	Assigned To
☐	Triggered	High	host01/file_exists : CheckFileExists CRITICAL - 0 files matched pattern /tmp/my-file.txt ⊕ SHOW DETAILS (1 triggered alert) #181	at 12:22 PM	API Service	Ops on-call

Resolve the alert in PagerDuty

To complete your workflow, restore the file that you removed so Sensu sends a resolution to PagerDuty:

```
touch /tmp/my-file.txt
```

In your PagerDuty account, the alert should now be listed under the "Resolved" tab.

To view the resolved event with sensuctl, run:

```
sensuctl event list
```

The response should show that the file is restored, with an OK (0) status:

```
  Entity      Check                          Output                        Status  Silenced      Timestamp
UUID
 ────────────── ───────────────
────────────────────────────────────────────────────────────────────── ────────
────────── ─────────────────────────── ───────────────────────────────────
 host01     file_exists   CheckFileExists OK - 1 files matched pattern /tmp/my-file.txt        0  false     2021-03-15
19:58:31 +0000 UTC   73f382fe-4290-48e8-955a-6efd54347b43
```

The web UI Events page will also show that the `file_check` check is passing.

Next steps

You should now have a working set-up with a `file_exists` check and a handler that sends alerts to

your PagerDuty account. To share and reuse the check and handler like code, save them to files and start building a monitoring as code repository.

You can customize your PagerDuty handler with configuration options like severity mapping, PagerDuty team-based routing and authentication via check and agent annotations, and event-based templating. Learn more about the Sensu PagerDuty integration and our curated, configurable quick-star template for incident management to integrate Sensu with your existing PagerDuty workflows.

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.

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 this check in place, follow Monitor server resources to add it.

Register the dynamic runtime asset

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

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

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

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-slack-handler:1.0.3
added asset: sensu/sensu-slack-handler:1.0.3

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-slack-handler"].
```

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

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

You should receive a confirmation message from sensuctl:

```
Created
```

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution. Read _the asset reference_ for more information about dynamic runtime asset builds.

Get a Slack webhook

If you're already the admin of a Slack, visit `https://YOUR_WORKSPACE_NAME_HERE.slack.com/services/new/incoming-webhook` and follow the steps to add the Incoming WebHooks integration, choose a channel, and save the settings. If you're not yet a Slack admin, create a new workspace and then create your webhook.

After saving, you can find your webhook URL under Integration Settings.

Create a handler

Use sensuctl to create a handler called `slack` that pipes observation data (events) to Slack using the `sensu-slack-handler` dynamic runtime asset. Edit the sensuctl command below to include your Slack webhook URL and the channel where you want to receive observation event data. For more information about customizing your Slack alerts, read the Sensu Slack Handler 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 receive a confirmation message:

```
Created
```

The `sensuctl handler create slack` command creates a handler resource. To view the `slack` handler definition, run:

SHELL

```
sensuctl handler info slack --format yaml
```

SHELL

```
sensuctl handler info slack --format wrapped-json
```

The `slack` handler resource definition will be similar to this example:

YML

```
---
type: Handler
api_version: core/v2
metadata:
  created_by: admin
  name: slack
  namespace: default
spec:
  command: sensu-slack-handler --channel '#monitoring'
  env_vars:
  - SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX
  filters: null
  handlers: null
  runtime_assets:
  - sensu-slack-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "slack",
    "namespace": "default"
  },
  "spec": {
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [
      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX"
    ],
    "filters": null,
    "handlers": null,
    "runtime_assets": [
      "sensu-slack-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

You can share and reuse this handler like code — save it to a file and start building a monitoring as code repository.

Assign the handler to a check

With the `slack` handler created, you can assign it to a check. To continue this example, use the `check_cpu` check created in Monitor server resources.

Assign your `slack` handler to the `check_cpu` check to receive Slack alerts when the CPU usage of your systems reaches the specific thresholds set in the check command:

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

To view the updated `check_cpu` resource definition, run:

SHELL

```shell
sensuctl check info check_cpu --format yaml
```

SHELL

```shell
sensuctl check info check_cpu --format wrapped-json
```

The updated check definition will be similar to this example:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: check_cpu
  namespace: default
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
  handlers:
  - slack
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
```

```
    stdin: false
    subdue: null
    subscriptions:
    - system
    timeout: 0
    ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "check_cpu",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "secrets": null,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "system"
    ],
    "timeout": 0,
```

```
      "ttl": 0
    }
  }
```

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 receive the following message in Slack:
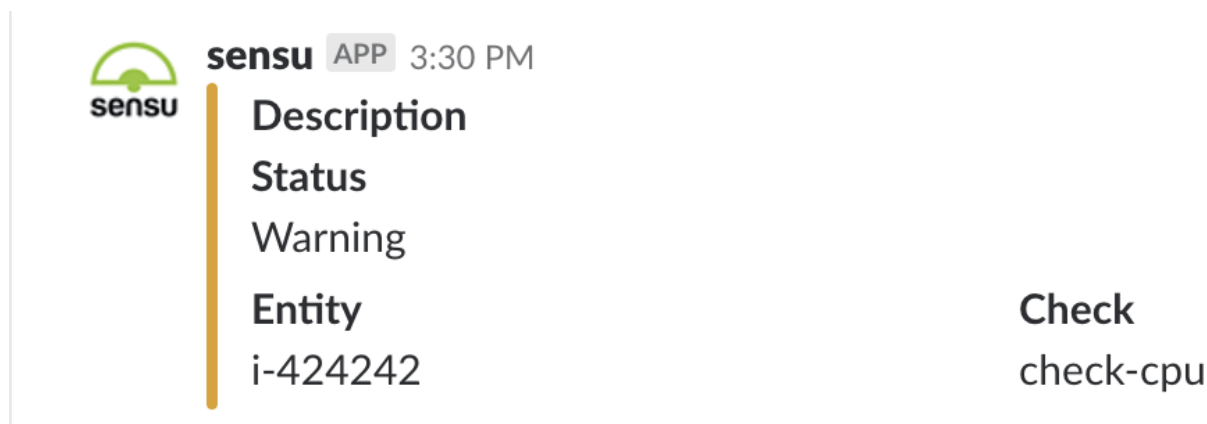
Verify the proper behavior of this handler with `sensu-backend` logs. Read Troubleshoot Sensu for log locations by platform.

Whenever an event is being handled, a log entry is added with the message `"handler":"slack","level":"debug","msg":"sending event to handler"`, followed by a second log entry with the message `"msg":"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.

Follow Send PagerDuty alerts with Sensu to configure a check that generates status events and a handler that sends Sensu alerts to PagerDuty for non-OK events.

Aggregate metrics with the Sensu StatsD listener

StatsD is a daemon, tool, and protocol that you can use to send, collect, and aggregate custom metrics Services that implement StatsD typically expose UDP port 8125 to receive metrics according to the line protocol `<metricname>:<value>|<type>` .

With StatsD, you can measure anything and everything. Collect custom metrics in your code and send them to a StatsD server to monitor applicaton performance. Monitor CPU, I/O, and network system levels with collection daemons. You can feed the metrics that StatsD aggregates to multiple different backends to store or visualize the data.

Use Sensu to implement StatsD

Sensu implements a StatsD listener on its agents. Each `sensu-agent` listens on the default port 8125 for UDP messages that follow the StatsD line protocol. StatsD aggregates the metrics, and Sensu translates them to Sensu metrics and events that can be passed to the event pipeline. You can configure the StatsD listener and access it with the netcat utility command:

```
echo 'abc.def.g:10|c' | nc -w1 -u localhost 8125
```

Metrics received through the StatsD listener are not stored in etcd. Instead, you must configure event handlers to send the data to a storage solution (for example, a time-series database like InfluxDB).

Configure the StatsD listener

Use 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

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.

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 dynamic runtime asset

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. This example uses the Sensu InfluxDB Handler dynamic runtime asset to power an `influx-db` handler.

Use `sensuctl asset add` to register the Sensu InfluxDB Handler dynamic runtime asset:

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

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu/sensu-influxdb-handler:3.1.2
added asset: sensu/sensu-influxdb-handler:3.1.2

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["influxdb-handler"].
```

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

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

Run `sensuctl asset list --format yaml` or `sensuctl asset list --format wrapped-json` to confirm that the dynamic runtime asset is ready to use.

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read *the asset reference* for more information about dynamic runtime asset builds.

Create the handler

Now that you have registered the dynamic runtime asset, you'll use sensuctl to create a handler called `influx-db` that pipes observation data (events) to InfluxDB with the `sensu-influxdb-handler` dynamic runtime asset.Edit the command below to include your database name, address, username, and password.For more information about the Sensu InfluxDB handler, read 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 receive a confirmation message:

```
Created
```

To review the complete resource definition for the handler resource you just created with sensuctl, run:

SHELL

```
sensuctl handler info influx-db --format yaml
```

```
sensuctl handler info influx-db --format wrapped-json
```

The `influx-db` handler resource definition will be similar to this example:

YML

```yaml
---
type: Handler
api_version: core/v2
metadata:
  created_by: admin
  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_PASS=password
  filters: null
  handlers: null
  runtime_assets:
  - influxdb-handler
  secrets: null
  timeout: 0
  type: pipe
```

JSON

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "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_PASS=password"
    ],
    "filters": null,
    "handlers": null,
    "runtime_assets": [
      "influxdb-handler"
    ],
    "secrets": null,
    "timeout": 0,
    "type": "pipe"
  }
}
```

You can share, reuse, and maintain this handler just like you would code: save it to a file and start building a monitoring as code repository.

Assign the InfluxDB handler to a check

With the `influx-db` handler created, you can assign it to a check for check output metric extraction. For example, if you followed Collect service metrics with Sensu checks, you created the check named `collect-metrics` .You can update the check's output metric format and output metric handlers to extract the metrics with your `influx-db` handler.

To update the output metric format, run:

```
sensuctl check set-output-metric-format collect-metrics influxdb_line
```

To update the output metric handlers, run:

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

You should receive a confirmation message:

```
Updated
```

To review the updated check resource definition, run:

SHELL

```
sensuctl check info collect-metrics --format yaml
```

SHELL

```
sensuctl check info collect-metrics --format wrapped-json
```

The updated `collect-metrics` check definition will be similar to this example:

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: collect-metrics
  namespace: default
spec:
  check_hooks: null
  command: metrics-disk-usage.rb
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: influxdb_line
  output_metric_handlers:
  - influx-db
  proxy_entity_name: ""
  publish: true
  round_robin: false
```

```
    runtime_assets:
    - sensu-plugins/sensu-plugins-disk-checks
    - sensu/sensu-ruby-runtime
    secrets: null
    stdin: false
    subdue: null
    subscriptions:
    - linux
    timeout: 0
    ttl: 0
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "collect-metrics",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "metrics-disk-usage.rb",
    "env_vars": null,
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "influxdb_line",
    "output_metric_handlers": [
      "influx-db"
    ],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-disk-checks",
      "sensu/sensu-ruby-runtime"
    ],
    "secrets": null,
```

```
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "linux"
      ],
      "timeout": 0,
      "ttl": 0
    }
  }
```

Assign the InfluxDB handler to the Sensu StatsD listener

To assign your `influx-db` handler to the Sensu StatsD listener at agent startup and pass all StatsD metrics into InfluxDB:

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

Validate the InfluxDB 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, metrics should start populating InfluxDB. You can verify proper handler behavior with `sensu-backend` logs. read Troubleshoot Sensu for log locations by platform.

Whenever an event is being handled, a log entry is added with the message `"handler":"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, read the handlers reference to learn more about using Sensu handlers.

Create handler templates

Sensu Go uses the Go template package, which allows you to generate text output that includes observation data from events. Sensu handler templates include HTML-formatted text and data derived from event attributes like `event.entity.name` and `.event.check.output`. This allows you to add meaningful, actionable context to alerts.

For example, a template for a brief Slack alert might include information about the affected entity and its status, as well as a link to the organization's playbook for resolving observability alerts:

```
<html>
The entity {{.Entity.Name}} has a status of {{.Check.State}}. The entity has
reported the same status for {{.Check.Occurrences}} preceding events.<br>
The playbook for managing this alert is available at
https://example.com/observability/alerts/playbook.
</html>
```

Template syntax and format

Handler templates use dot notation syntax to access event attributes, with the event attribute wrapped in double curly braces. The initial dot indicates `event`.

For example, in a handler template, a reference to the event attribute `.Check.occurrences` becomes `{{.Check.Occurrences}}`.

Use HTML to format the text and spacing in your templates. All text outside double curly braces is copied directly into the template output, with HTML formatting applied.

Available event attributes

If you are using a plugin that supports template output, every attribute in the Sensu event is available. However, the attribute capitalization pattern is different for handler templates than for event format.

The table below lists the event attributes that are available to use in handler templates, in the correct dot notation and capitalization pattern. You can also use the <u>template toolkit command</u> to print available event attributes for a specific event.

attribute	attribute	attribute
`.HasCheck`	`.HasMetrics`	`.IsIncident`
`.IsResolution`	`.IsSilenced`	`.Timestamp`
`.Check.Annotations`	`.Check.CheckHooks`	`.Check.Command`
`.Check.Cron`	`.Check.DiscardOutput`	`.Check.Duration`
`.Check.EnvVars`	`.Check.Executed`	`.Check.ExtendedAttributes`
`.Check.Handlers`	`.Check.HighFlapThreshold`	`.Check.History`
`.Check.Hooks`	`.Check.Interval`	`.Check.Issued`
`.Check.Labels`	`.Check.LastOK`	`.Check.LowFlapThreshold`
`.Check.MaxOutputSize`	`.Check.Name`	`.Check.Namespace`
`.Check.Occurrences`	`.Check.OccurrencesWatermark`	`.Check.Output`
`.Check.OutputMetricFormat`	`.Check.OutputMetricHandlers`	`.Check.ProxyEntityName`
`.Check.ProxyRequests`	`.Check.Publish`	`.Check.RoundRobin`
`.Check.RuntimeAssets`	`.Check.Secrets`	`.Check.Silenced`
`.Check.State`	`.Check.Status`	`.Check.Stdin`

`.Check.Subdue`	`.Check.Subscriptions`	`.Check.Timeout`
`.Check.TotalState Change`	`.Check.Ttl`	`.Entity.Annotations`
`.Entity.Deregiste r`	`.Entity.Deregistration`	`.Entity.EntityClass`
`.Entity.ExtendedA ttributes`	`.Entity.KeepaliveHandlers`	`.Entity.Labels`
`.Entity.LastSeen`	`.Entity.Name`	`.Entity.Namespace`
`.Entity.Redact`	`.Entity.SensuAgentVersion`	`.Entity.Subscriptions`
`.Entity.System`	`.Entity.System.Arch`	`.Entity.System.ARMVersion`
`.Entity.System.Cl oudProvider`	`.Entity.System.Hostname`	`.Entity.System.LibcType`
`.Entity.System.Ne twork`	`.Entity.System.OS`	`.Entity.System.Platform`
`.Entity.System.Pl atformFamily`	`.Entity.System.PlatformVersio n`	`.Entity.System.Processes`
`.Entity.System.VM Role`	`.Entity.System.VMSystem`	`.Entity.User`
`.Metrics.Handler s`	`.Metrics.Points`	

Template toolkit command

The Sensu template toolkit command is a sensuctl command plugin you can use to print a list of available event attributes in handler template dot notation syntax and validate your handler template output.

The template toolkit command uses event data you supply via STDIN in JSON format.

Add the Sensu template toolkit command asset to Sensu:

```
sensuctl asset add sensu/template-toolkit-command:0.4.0 -r template-toolkit-command
```

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

You can also download the latest asset definition from Bonsai.

Run `sensuctl asset list` to confirm that the asset is ready to use:

```
      Name                              URL                          Hash

 _____

_____
_____ _____

  template-toolkit-command  //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_windows_amd64.tar.gz
019ccf3
  template-toolkit-command  //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_darwin_amd64.tar.gz
b771813
  template-toolkit-command  //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_armv7.tar.gz
4e7ad65
  template-toolkit-command  //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_arm64.tar.gz
02eca1f
  template-toolkit-command  //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_386.tar.gz          56ed603
  template-toolkit-command  //assets.bonsai.sensu.io/.../template-toolkit-command_0.4.0_linux_amd64.tar.gz
7dbd2c6
```

Print available event attributes

Use the template toolkit command to print a list of the available event attributes as well as the correct dot notation and capitalization pattern for a specific event (in this example, `event.json`):

```
cat event.json | sensuctl command exec template-toolkit-command -- --dump-names
```

The response lists the available attributes for the event:

```
INFO[0000] asset includes builds, using builds instead of asset    asset=template-
toolkit-command component=asset-manager entity=sensuctl
.Event{
    .Timestamp: 1580310179,
    .Entity{
        .EntityClass: "agent",
        .System:       .System{
    [...]
    .Check{
        .Command:              "",
        .Handlers:             {"keepalive"},
        .HighFlapThreshold: 0x0,
    [...]
```

In this example, the response lists the available event attributes `.Timestamp`,
`.Entity.EntityClass`, `.Entity.System`, `.Check.Command`, `.Check.Handlers`, and
`.Check.HighFlapThreshold`.

You can also use `sensuctl event info <entity_name> <check_name>` to print the correct notation
and pattern: template output for a specific event (in this example, an event for entity `server01` and
check `server-health`):

```
sensuctl event info server01 server-health --format json | sensuctl command exec
template-toolkit-command -- --dump-names
```

The response lists the available attributes for the event:

```
INFO[0000] asset includes builds, using builds instead of asset    asset=template-
toolkit-command component=asset-manager entity=sensuctl
.Event{
    .Timestamp: 1580310179,
    .Entity:{
        .EntityClass:          "proxy",
        .System:               .System{
    [...]
    .Check:{
        .Command:              "health.sh",
        .Handlers:             {"slack"},
```

```
        .HighFlapThreshold: 0x0,
    [...]
```

Validate handler template output

Use the template toolkit command to validate the dot notation syntax and output for any event attribute.

For example, to test the output for the `{{.Check.Name}}` attribute for the event `event.json`:

```
cat event.json | sensuctl command exec template-toolkit-command -- --template "
{{.Check.Name}}"
```

The response will list the template output:

```
INFO[0000] asset includes builds, using builds instead of asset  asset=template-
toolkit-command component=asset-manager entity=sensuctl
executing command with --template {{.Check.Name}}
Template String Output: keepalive
```

In this example, the command validates that for the `event.json` event, the handler template will replace `{{.Check.Name}}` with `keepalive` in template output.

You can also use `sensuctl event info <entity_name> <check_name>` to validate template output for a specific event (in this example, an event for entity `webserver01` and check `check-http`):

```
sensuctl event info webserver01 check-http --format json | sensuctl command exec
template-toolkit-command -- --template "Server: {{.Entity.Name}} Check:
{{.Check.Name}} Status: {{.Check.State}}"
```

The response will list the template output:

```
Executing command with --template Server: {{.Entity.Name}} Check: {{.Check.Name}}
Status: {{.Check.State}}
Template String Output: Server: "webserver01 Check: check-http Status: passing"
```

Sensu Email Handler plugin

The Sensu Email Handler plugin allows you to provide a template for the body of the email. For example, this template will produce an email body that includes the name of the check and entity associated with the event, the status and number of occurrences, and other event details:

```
<html>
Greetings,

<h3>Informational Details</h3>
<b>Check</b>: {{.Check.Name}}<br>
<b>Entity</b>: {{.Entity.Name}}<br>
<b>State</b>: {{.Check.State}}<br>
<b>Occurrences</b>: {{.Check.Occurrences}}<br>
<b>Playbook</b>: https://example.com/monitoring/wiki/playbook
<h3>Check Output Details</h3>
<b>Check Output</b>: {{.Check.Output}}
<h4>Check Hook(s)</h4>
{{range .Check.Hooks}}<b>Hook Name</b>:  {{.Name}}<br>
<b>Hook Command</b>:  {{.Command}}<br>
<b>Hook Output</b>: {{.Output}}<br>
{{end}}<br>
<br>
<br>
#monitoringlove,<br>
<br>
Sensu<br>
</html>
```

The Sensu Email Handler plugin also includes a UnixTime function that allows you to print timestamp values from events in human-readable format. Read the Sensu Email Handler Bonsai page for details.

Sensu PagerDuty Handler example

The Sensu PagerDuty Handler plugin includes a basic template for the PagerDuty alert summary:

```
"{{.Entity.Name}}/{{.Check.Name}} : {{.Check.Output}}"
```

With this template, the summary for every alert in PagerDuty will include:

- ¬ The name of the affected entity.

- ¬ The name of the check that produced the event.

- ¬ The check output for the event.

Read the Sensu PagerDuty Handler Bonsai page for details.

Silencing reference

Sensu's silencing capability allows you to suppress event handler execution on an ad hoc basis so you can plan maintenance and reduce alert fatigue.Silences are created on an ad hoc basis using `sensuctl` .

Successfully created silencing entries are assigned a `name` in the format `$SUBSCRIPTION:$CHECK` , where `$SUBSCRIPTION` is the name of a Sensu entity subscription and `$CHECK` is the name of a Sensu check.You can use silences to silence checks on specific entities by taking advantage of per-entity subscriptions (for example, `entity:$ENTITY_NAME`).

When creating a silencing entry, you can specify a combination of checks and subscriptions, but only one or the other is strictly required.For example, if you create a silencing entry specifying only a check, its name will contain an asterisk (or wildcard) in the `$SUBSCRIPTION` position.This indicates that any event with a matching check name will be marked as silenced, regardless of the originating entities' subscriptions.

Conversely, a silencing entry that specifies only a subscription will have a name with an asterisk in the `$CHECK` position.This indicates that any event where the originating entities' subscriptions match the subscription specified in the entry will be marked as silenced, regardless of the check name.

These silences are persisted in the Sensu datastore.When the Sensu server processes subsequent check results, it retrieves matching silences from the store.If there are one or more matching entries, the event is updated with a list of silenced entry names.When the check name or subscription described in a silencing entry matches an event, the event will include the `silenced` attribute, which lists the silencing entries that match the event.

Silenced checks still create events, and events from silenced checks are still passed to handlers.To prevent handler execution for events from silenced checks, make sure the handler definition includes the built-in `not_silenced` event filter.The `not_silenced` event filter prevents handlers from processing events that include the `silenced` attribute.

Silencing examples

This example shows a silencing resource definition that uses a per-entity subscription to silence any alerts on a single Sensu entity, `i-424242` :

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  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

```
{
  "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
  }
}
```

The following example shows how to silence a check named `check_ntp` on entity `i-424242`, ensuring the silencing entry is deleted after the underlying issue is resolved:

YML

```yml
---
type: Silenced
api_version: core/v2
metadata:
  name: entity:i-424242:check_ntp
  namespace: default
  labels:
  annotations:
spec:
  subscription: entity:i-424242
  check: check_ntp
  expire_on_resolve: true
```

JSON

```json
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "entity:i-424242:check_ntp",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "subscription": "entity:i-424242",
    "check": "check_ntp",
    "expire_on_resolve": true
  }
}
```

The optional `expire_on_resolve` attribute used in this example indicates that when the server processes a matching check from the specified entity with status OK, the silencing entry will be removed automatically.

When used in combination with other attributes (like `creator` and `reason`), this gives Sensu operators a way to acknowledge that they received an alert, suppress additional notifications, and automatically clear the silencing entry when the check status returns to normal.

Silencing specification

Silenced entry names

Silences must contain either a subscription or check name and are identified by the combination of `$SUBSCRIPTION:$CHECK` . If a check or subscription is not provided, it will be substituted with a wildcard (asterisk): `$SUBSCRIPTION:*` or `*:$CHECK` .

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Silences should always be type `Silenced` .
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	String
| example | **YML**<br><br>```yaml
type: Silenced
```<br><br>**JSON**<br><br>```json
{
  "type": "Silenced"
}
``` |

| api_version | |

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

YML

```
api_version: core/v2
```

JSON

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

metadata

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

YML

```
metadata:
  name: appserver:mysql_status
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
```

```
JSON

{
  "metadata": {
    "name": "appserver:mysql_status",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "region": "us-west-1"
    }
  }
}
```

spec	
description	Top-level map that includes the silencing entry spec attributes.
required	Required for silences in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs
example	YML

```
spec:
  expire: -1
  expire_on_resolve: false
  creator: admin
  reason:
  check:
  subscription: entity:i-424242
  begin: 1542671205
```

```
JSON

{
  "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	**YML**

```
  name: appserver:mysql_status
```

JSON

```
  {
    "name": "appserver:mysql_status"
  }
```

namespace	
description	Sensu RBAC namespace that the silencing entry belongs to.

required	false
type	String
default	`default`

example	**YML**

```
namespace: production
```

JSON

```
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the silence or last updated the silence. Sensu automatically populates the `created_by` field when the silence is created or updated.
required	false
type	String

example	**YML**

```
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

labels

description	Custom attributes to include with observation data in events that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example	**YML**
	
	```yml\nlabels:\n environment: development\n region: us-west-2\n```
	
	JSON
	
	```json\n{\n "labels": {\n "environment": "development",\n "region": "us-west-2"\n }\n}\n```

annotations

description	Non-identifying metadata to include with observation data in events that you can access with event filters. You can use annotations to add data

that's meaningful to people or external tools that interact with Sensu.

In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.

required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
| example | **YML**<br><br>```yaml
annotations:
  managed-by: ops
  playbook: www.example.url
```<br><br>**JSON**<br><br>```json
{
  "annotations": {
    "managed-by": "ops",
    "playbook": "www.example.url"
  }
}
``` |

Spec attributes

check	
description	Name of the check the entry should match.
required	true, unless `subscription` is provided
type	String
| example | **YML**<br><br>```yaml
check: haproxy_status
``` |

JSON

```json
{
  "check": "haproxy_status"
}
```

subscription

description	Name of the subscription the entry should match.
required	true, unless `check` is provided
type	String

YML

example

```yaml
subscription: entity:i-424242
```

JSON

```json
{
  "subscription": "entity:i-424242"
}
```

begin

description	Time at which silence entry goes into effect. In epoch.
required	false
type	Integer

YML

example

```yaml
begin: 1512512023
```

JSON

```json
{
  "begin": 1512512023
}
```

expire

description	Number of seconds until the entry should be deleted.
required	false
type	Integer
default	-1

YML

```yml
expire: 3600
```

JSON

```json
{
  "expire": 3600
}
```

example

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	**YML**
| | ```yaml
expire_on_resolve: true
``` |
| | **JSON** |
| | ```json
{
  "expire_on_resolve": true
}
``` |

creator

description	Person, application, or entity responsible for creating the entry.
required	false
type	String
default	null
example	**YML**
| | ```yaml
creator: Application Deploy Tool 5.0
``` |
| | **JSON** |
| | ```json
{
  "creator": "Application Deploy Tool 5.0"
}
``` |

reason

description	Explanation of the reason for creating the entry.
required	false

type	String
default	null

example	

```yaml
reason: rebooting the world
```

JSON

```json
{
  "reason": "rebooting the world"
}
```

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

```yaml
---
type: Silenced
api_version: core/v2
metadata:
  name: appserver
  namespace: default
  labels:
  annotations:
spec:
  subscription: appserver
```

JSON

```json
{
  "type": "Silenced",
```

```
    "api_version": "core/v2",
    "metadata": {
      "name": "appserver",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "spec": {
      "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

```
---
type: Silenced
api_version: core/v2
metadata:
  name: appserver:mysql_status
  namespace: default
  labels:
  annotations:
spec:
  subscription: appserver
  check: mysql_status
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "appserver:mysql_status",
```

```
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "spec": {
      "subscription": "appserver",
      "check": "mysql_status"
    }
  }
```

Silence a specific check on every entity

To silence the check `mysql_status` on every entity in your infrastructure, regardless of subscriptions, you only need to provide the check name:

YML

```
---
type: Silenced
api_version: core/v2
metadata:
  name: mysql_status
  namespace: default
  labels:
  annotations:
spec:
  check: mysql_status
```

JSON

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "mysql_status",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
```

```
  "spec": {
    "check": "mysql_status"
  }
}
```

Delete a silence

To delete a silencing entry, you must provide its name.

Subscription-only silencing entry names will contain an asterisk (or wildcard) in the `$SUBSCRIPTION` position, similar to this example:

YML

```yml
name: appserver:*
```

JSON

```json
{
  "name": "appserver:*"
}
```

Check-only silencing entry names will contain an asterisk (or wildcard) in the `$CHECK` position, similar to this example:

YML

```yml
name: '*:mysql_status'
```

JSON

```json
{
  "name": "*:mysql_status"
}
```

Plan maintenance windows with silencing

As the Sensu backend processes check results, the server executes <u>event handlers</u> to send alerts to personnel or otherwise relay observation data (events) 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:

- <u>Silence all checks on a specific entity</u>
- <u>Silence a specific check on a specific entity</u>
- <u>Silence all checks on entities with a specific subscription</u>
- <u>Silence a specific check on entities with a specific subscription</u>
- <u>Silence a specific check on every entity</u>

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 '2021-03-14 01:00:00 -04:00' \
```

```
  --expire 3600 \
  --reason 'Server upgrade'
```

This command creates the following silenced resource definition:

YML

```yaml
type: Silenced
api_version: core/v2
metadata:
  created_by: admin
  name: entity:i-424242:check-http
  namespace: default
spec:
  begin: 1615698000
  check: check-http
  creator: admin
  expire: 3600
  expire_at: 1615701600
  expire_on_resolve: false
  reason: Server upgrade
  subscription: entity:i-424242
```

JSON

```json
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "entity:i-424242:check-http",
    "namespace": "default"
  },
  "spec": {
    "begin": 1615698000,
    "check": "check-http",
    "creator": "admin",
    "expire": 3600,
    "expire_at": 1615701600,
    "expire_on_resolve": false,
    "reason": "Server upgrade",
```

```
      "subscription": "entity:i-424242"
    }
  }
}
```

Read 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:

SHELL

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

SHELL

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

The response will list the silenced resource definition.

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

```
   Entity      Check     Output    Status   Silenced      Timestamp
 ─────────── ─────────── ─────────── ────────── ──────────────────────
 ──────────────────────────────
  i-424242    check-http            0       true    2021-03-14 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

Read the silencing reference for in-depth documentation about silenced entries.

Operations

The Operations category will help you get Sensu up and running, from your first installation in your local development environment through a large-scale Sensu deployment using secrets management. You'll also learn how to keep your Sensu implementation running, with guides for upgrading, monitoring, and troubleshooting.

Monitoring as code

Monitoring as code with Sensu explains how to use Sensu's end-to-end monitoring as code approach to manage your observability configuration the same way you build, test, and deploy your applications and infrastructure.

Deploy Sensu

Deploy Sensu describes how to plan, install, configure, and deploy Sensu's flexible monitoring and observability pipeline.

To plan your Sensu deployment, read the hardware requirements and deployment architecture pages. To start using Sensu locally or in development environments, follow the steps in the Install Sensu guide.

When you're ready to deploy Sensu in production, learn to generate certificates, secure your Sensu installation, run a Sensu cluster, and reach multi-cluster visibility. You'll also find guides for scaling your implementation with Sensu's Enterprise datastore and using configuration management tools to ensure repeatable Sensu deployments and consistent configuration.

Control Access

Control Access explains how Sensu administrators control access by authentication (verifying user identities) and authorization (establishing and managing user permissions for Sensu resources).

Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool. Use Sensu's built-in basic authentication provider or configure external

authentication providers via Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) to authenticate your Sensu users.

Next, learn to configure authorization for your authenticated Sensu users with role-based access control (RBAC) and set up user permissions for interacting with Sensu resources.

Maintain Sensu

Maintain Sensu includes upgrade, migration, troubleshooting, and license management information to keep your Sensu implementation running smoothly.

Follow our step-by-step instructions to upgrade to the latest version of Sensu from any earlier version. If you're still using Sensu Core or Sensu Enterprise, read Migrate from Sensu Core and Sensu Enterprise to Sensu Go. You can also learn how to activate and and view your commercial Sensu license or troubleshoot to identify and resolve problems with your Sensu implementation, from reading and configuring Sensu service logs to using Sensu handlers and filters to test and debug your implementation.

Monitor Sensu

Monitor Sensu covers how to log Sensu services, monitor your Sensu backend with a secondary instance, and retrieve and process health data for your Sensu cluster. You can also learn about Tessen the Sensu call-home service, which helps us understand how Sensu is being used and make informed decisions about product improvements.

Manage Secrets

Manage Secrets explains how to use Sensu's secrets management to eliminate the need to expose secrets like usernames, passwords, and access keys in your Sensu configuration. Learn to configure secrets and secrets providers resources to obtain secrets from one or more external secrets providers, refer to external secrets, and consume secrets via backend environment variables.

Monitoring as code with Sensu

Sensu's end-to-end monitoring as code approach allows you to manage your observability configuration the same way you build, test, and deploy your applications and infrastructure, like Kubernetes and Terraform. Monitoring as code combines composable building blocks with robust APIs so you can define your entire observability configuration as declarative YAML or JSON code and improve visibility, reliability, portability, and repeatability.

When a new endpoint starts up, like a cloud compute instance or Kubernetes Pod, Sensu automatically registers itself with the platform and starts collecting observability data according to the code in your configuration files. Teams can share and remix observability configurations for collecting events and metrics, diagnosing issues, sending alerts, and automatically remediating problems.

- Share, edit, review, and version your observability configuration files just like you would with other "as code" solutions, within one team or among teams across your organization.

- Maintain revision control and change history for your observability configurations.

- Export the Sensu configuration for one environment and replicate the same configuration in other environments.

- Remove, restore, back up, and recover Sensu instances based on your Sensu configuration files.

- Include your observability configuration in your centralized continuous integration/continuous delivery (CI/CD) pipeline to keep your configuration files aligned with your product and services.

To get started with monitoring as code, you'll need a repository and configuration files that contain your resource definitions.

Create a monitoring as code repository

Create a monitoring as code repository for the configuration files that contain the Sensu resource definitions you use for monitoring and observability. You can use any source control repository.

The way you will use your configuration files will help you choose the best structure for your monitoring as code repository. For example, if you are likely to share observability components or manage your configuration files as part of your CI/CD workflow, it probably makes sense to use individual

configuration files for different types of resources: one file for all of your checks, one file for all of your handlers, and so on. If you want to facilitate more granular sharing, you can save one resource definition per file.

If you want to share complete end-to-end observability configurations with your colleagues, you might save all of the resource definitions for each observability configuration in a single configuration file. This allows others to read through an entire configuration without interruption, and it's convenient for demonstrating a complete Sensu configuration. However, a single configuration file that includes every resource type isn't the best structure for CI/CD management or sharing resources among teams.

SensuFlow, our GitHub Action for managing Sensu resources via repository commits, requires a repository structure organized by clusters and namespaces. All resources of each type for each namespace are saved in a single configuration file:

```
.sensu/
  cluster/
    namespaces.yml
  namespaces/
    <namespace>/
      checks/
      hooks/
      filters/
      handlers/
      handlersets/
      mutators/
```

Adopt a configuration file strategy

Configuration files contain your Sensu resource definitions. You can build configuration files as you go, adding resource definitions as you create them. You can also create your entire observability configuration first, then export some or all of your resource definitions to a file. Or, you can use a mix: export all of your existing resource definitions to configuration files and append new resources as you create them.

When you are ready to replicate your exported resource definitions, use `sensuctl create` .

NOTE: You cannot replicate API key or user resources from a `sensuctl dump` *export.*

> *API keys must be reissued, but you can use your exported configuration file as a reference for granting new API keys to replace the exported keys.*
>
> *When you export users, required password attributes are not included. You must add a `password_hash` or `password` to `users` resources before replicating them with the `sensuctl create` command.*

Build as you go

To build as you go, use sensuctl commands to retrieve your Sensu resource definitions as you create them and copy the definitions into your configuration files.

For example, if you follow Monitor server resources and create a check named `check_cpu`, you can retrieve that check definition in YAML or JSON format:

SHELL

```
sensuctl check info check_cpu --format yaml
```

SHELL

```
sensuctl check info check_cpu --format wrapped-json
```

The sensuctl response will include the complete `check_cpu` resource definition in the specified format:

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  name: check_cpu
  namespace: default
spec:
  check_hooks: null
  command: check-cpu-usage -w 75 -c 90
  env_vars: null
```

```
  handlers:
  - slack
  high_flap_threshold: 0
  interval: 60
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  proxy_entity_name: ""
  publish: true
  round_robin: false
  runtime_assets:
  - check-cpu-usage
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - system
  timeout: 0
  ttl: 0
```

JSON

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "check_cpu",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu-usage -w 75 -c 90",
    "env_vars": null,
    "handlers": [
      "slack"
    ],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "output_metric_format": "",
    "output_metric_handlers": null,
```

```
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "secrets": null,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "system"
      ],
      "timeout": 0,
      "ttl": 0
    }
  }
```

You can copy these resource definitions and paste them into manually created configuration files located anywhere on your system.

Alternatively, you can view resource definitions and copy them into a new or existing configuration file with a single sensuctl command. To use the following examples, replace `<resource>` with the resource type (like `check`) and replace `<resource_name>` with the name of the resource (like `check_cpu`).

¬ Copy the resource defintion to a new file (or overwrite an existing file with the same name):

SHELL

```
sensuctl <resource> info <resource_name> --format yaml > resource.yml
```

SHELL

```
sensuctl <resource> info <resource_name> --format wrapped-json >
resource.json
```

¬ Copy the resource defintion to a new file (or overwrite an existing file with the same name) and show the resource definition in STDOUT:

```
sensuctl <resource> info <resource_name> --format yaml | tee resource.yml
```

```
sensuctl <resource> info <resource_name> --format wrapped-json | tee
resource.json
```

¬ Append the resource defintion to an existing file:

```
sensuctl <resource> info <resource_name> --format yaml >> resource.yml
```

```
sensuctl <resource> info <resource_name> --format wrapped-json >>
resource.json
```

¬ Append the resource defintion to an existing file and show the resource definition in STDOUT:

```
sensuctl <resource> info <resource_name> --format yaml | tee -a resource.yml
```

```
sensuctl <resource> info <resource_name> --format wrapped-json | tee -a
resource.json
```

Export existing resources

If you've already created observability resources, use `sensuctl dump` to create a copy of your existing resource definitions.

First, create a sensu directory:

```
mkdir sensu
```

Then, copy your observability resource definitions according to the repository structure you are using. For example, if you want to save resources according to type and namespace, this command will save all of your check definitions for the `production` namespace in one configuration file:

SHELL

```
sensuctl dump core/v2.CheckConfig \
--namespace production \
--format yaml | > sensu/namespaces/production/checks.yml
```

SHELL

```
sensuctl dump core/v2.CheckConfig \
--namespace production \
--format wrapped-json | > sensu/namespaces/production/checks.json
```

Repeat this command for each resource type in each of your namespaces.

Strip namespaces from resource definitions

To replicate and reuse resources in any namespace without manual editing, create a copy of your existing resources with the namespaces stripped from their definitions:

SHELL

```
sensuctl dump all \
--all-namespaces \
--format yaml | grep -v "^\s*namespace:" > sensu/resources.yml
```

SHELL

```
sensuctl dump all \
--all-namespaces \
```

```
--format wrapped-json | grep -v "^\s*namespace:" > sensu/resources.json
```

Best practices for monitoring as code

The monitoring as code approach is flexible — you can use any source control repository and choose your own directory structure — but following a few best practices will contribute to a successful implementation.

- ⌐ To maintain consistency, save all of your resources as only one file type: YAML or JSON.

- ⌐ Include all dependencies within a resource definition. For example, if a handler requires a dynamic runtime asset and a secret, include the asset and secret definitions with the definition for the handler itself.

- ⌐ Choose the labels you use in your resource definitions with care. CI/CD systems like SensuFlow rely on labels to determine which resources to delete, so if all of your resources have the same labels, you could delete resources you didn't intend to be managed in a particular CI/CD workflow.

- ⌐ Establish a resource-labeling schema throughout your organization to facilitate CI/CD. Following the same method for applying labels helps keep unmanaged Sensu resources from multiplying and allows different teams to confidently deploy their own CI/CD workflows without the risk of accidentally deleting another team's resources.

Implement CI/CD with monitoring as code

When you're ready, you can expand your monitoring as code practices to include managing your Sensu configuration files with a CI/CD workflow. CI/CD takes the manual work out of maintaining and updating your monitoring as code repository so that any updates to the Sensu resources in your monitoring as code repository are reflected in your Sensu configuration in a timely manner.

If you're already using CI/CD, you already have workflows for versioning, building, testing, and deploying your code. Integrating monitoring as code means your monitoring can go through the same CI/CD workflows as your code.

There's no one "correct" way to implement CI/CD with monitoring as code, but the SensuFlow GitHub Action offers a turnkey reference implementation that helps you create your own monitoring as code workflow and start managing Sensu resources via repository commits.

Use SensuFlow for CI/CD monitoring as code

SensuFlow is a git-based, prescriptive monitoring as code workflow that uses <u>sensuctl</u> (including <u>sensuctl prune</u>) to synchronize your monitoring and observability code with your Sensu deployments.

> **NOTE**: *SensuFlow is available for technical preview, and individual components in the workflow may change.Before you use SensuFlow in production, test it in a development environment or a dedicated test namespace in your current environment.*

SensuFlow requires:

- ¬ A code repository of Sensu resource definitions

- ¬ A Sensu <u>role-based access control (RBAC)</u> service account with permission to manage all resources in your repository

- ¬ A resource labeling convention to designate which resources the SensuFlow workflow should manage

- ¬ Integration with your CI/CD system to run sensuctl commands as the service account user from the repository of resource definitions

Read the <u>SensuFlow GitHub Action marketplace page</u> and <u>Monitoring as code with Sensu Go and SensuFlow</u> to use SensuFlow as your monitoring as code workflow.

Deploy Sensu

Use the information and instructions in the Deploy Sensu category to plan, install, configure, and deploy Sensu's flexible monitoring and observability pipeline.

Plan your Sensu deployment

Find Sensu agent and backend requirements and networking and cloud recommendations in the hardware requirements.

Deployment architecture for Sensu describes planning considerations and recommendations for a production-ready Sensu deployment, along with communication security details and diagrams showing single, clustered, and large-scale deployment architectures.

Install Sensu

When you're ready to start using Sensu, the pathway you follow will depend on your monitoring and observability needs.No matter which pathway you choose, you should begin with the Install Sensu guide.If you just want to use Sensu locally, you can do that by installing Sensu according to the steps in the guide.You can also use the Install Sensu guide to set up proof-of-concept and testing in a development environment.

Deploy Sensu in production

To deploy Sensu for use outside of a local development environment, install Sensu and follow these guides to achieve a production-ready installation:

1. Generate certificates, which you will need to secure a Sensu cluster and its agents.
2. Secure your Sensu installation using the certificates you generate to make Sensu production-ready.
3. Run a Sensu cluster, a group of three or more sensu-backend nodes connected to a shared database, to improve Sensu's availability, reliability, and durability.
4. Reach multi-cluster visibility with federation so you can gain visibility into the health of your infrastructure and services across multiple distinct Sensu instances within a single web UI and

mirror your changes in one cluster to follower clusters.

Read the etcd replicators reference to learn how the etcd-replicators datatype in the federation API allows you to manage role-based access control (RBAC) resources in one place and mirror your changes to follower clusters.

Scale your Sensu implementation

As the number of entities and checks in your Sensu implementation grows, so does the rate of events being written to the datastore. In clustered etcd deployments, each event must be replicated to each cluster member, which increases network and disk IO utilization.

Sensu's Enterprise datastore allows you to configure an external PostgreSQL instance for event storage so you can scale your monitoring and observability workflows beyond etcd's 8GB limit. Scale your Sensu implementation to many thousands of events per second, achieve much higher rates of event processing, and minimize the replication communication between etcd peers.

Read the datastore reference for the Enterprise datastore requirements and specifications.

For deployments at scale, configuration management tools can help ensure repeatable Sensu deployments and consistent configuration among Sensu backends. Ansible, Chef, and Puppet have well-defined Sensu modules to help you get started.

Hardware requirements

Sensu backend requirements

Backend minimum requirements

This configuration is the minimum required to run the Sensu backend (although it is insufficient for production use):

- 64-bit Intel or AMD CPU

- 4 GB RAM

- 4 GB free disk space

- 10 mbps network link

Review the backend recommended configuration for production recommendations.

Backend recommended configuration

This 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, ARM (ARMv5 minimum), or MIPS CPU

- ⌐ 128 MB RAM

- ⌐ 10 mbps network link

Review 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 the backend logs include WebSocket timeouts, 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).

Install Sensu

This installation guide describes how to install the Sensu backend, Sensu agent, and sensuctl command line tool.If you're trying Sensu for the first time, we recommend setting up a local environment using the Sensu sandbox.

> **NOTE**: *The instructions in this guide explain how to install Sensu for proof-of-concept purposes or testing in a development environment.If you will deploy Sensu to your infrastructure, we recommend one of our supported packages, Docker images, or configuration management integrations, as well as securing your installation with transport layer security (TLS).Read Generate certificates next to get the certificates you will need for TLS.*

Sensu downloads are provided under the Sensu commercial license.

Sensu Go is packaged for Linux, Windows (agent and CLI only), macOS (CLI only), and Docker. Review supported platforms for more information.

Architecture overview

Sensu works differently from other monitoring and observability solutions.Instead of provisioning each device, server, container, or sidecar you want to monitor, you install the Sensu agent on each infrastructure component.

Sensu agents are lightweight clients that run on the infrastructure components you want to monitor. Agents are responsible for creating status and metric events to send to the Sensu backend event pipeline.Agents automatically register with Sensu as entities when you start them up and connect to the Sensu backend with no need for further provisioning.You only need to specify the IP address for the Sensu backend server — you do not need to list the components to monitor in the backend.

The **Sensu backend** is powered by an an embedded transport and etcd datastore.The backend sends specific checks for each agent to execute according to the subscriptions you define in the agent configuration.Sensu automatically downloads the files needed to run the checks from an asset repository like Bonsai or a local repo and schedules the checks on each agent.The agents execute the checks the backend sends to their subscriptions and send the resulting status and metric events to the backend event pipeline, which gives you flexible, automated workflows to route these events.
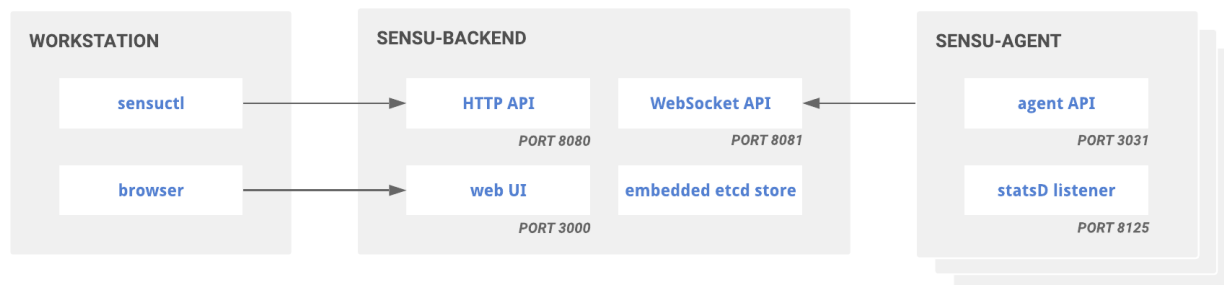
The Sensu backend keeps track of all self-registered agents. If the backend loses a keepalive signal from any of the agents, it flags the agent and generates an event. You can configure your instance so that when an agent (for example, a server) shuts down gracefully, the agent automatically de-registers from the backend and does not generate an alert.

Sensu backends require persistent storage for their embedded database, disk space for local asset caching, and several exposed ports. Agents that use Sensu dynamic runtime assets require some disk space for a local cache.

For more information, read Secure Sensu. Read deployment architecture and hardware requirements for deployment recommendations.

Ports

Sensu backends require the following ports:

Port	Protocol	Description
2379	gRPC	Sensu storage client: Required for Sensu backends using an external etcd instance
2380	gRPC	Sensu storage peer: Required for other Sensu backends in a cluster
3000	HTTP/HTTPS	Sensu web UI: Required for all Sensu backends using a Sensu web UI
6060	HTTP/HTTPS	Required for all Sensu backends when performance profiling is enabled via debug setting
8080	HTTP/HTTPS	Sensu API: Required for all users accessing the Sensu API

Port	Protocol	Description
8081	WS/WSS	Agent API: Required for all Sensu agents connecting to a Sensu backend

The Sensu agent uses the following ports:

Port	Protocol	Description
3030	TCP/UDP	Sensu agent socket: Required for Sensu agents using the agent socket
3031	HTTP	Sensu agent API: Required for all users accessing the agent API
8125	UDP	StatsD listener: Required for all Sensu agents using the StatsD listener

The agent TCP and UDP sockets are deprecated in favor of the agent API.

Install the Sensu backend

The Sensu backend is available for Ubuntu/Debian, RHEL/CentOS, and Docker.Review supported platforms for more information.

1. Download
DOCKER

```
# All Sensu Docker images contain a Sensu backend and a Sensu agent

# Pull the Alpine-based image
docker pull sensu/sensu

# Pull the image based on Red Hat Enterprise Linux
docker pull sensu/sensu-rhel
```

SHELL

```
# Add the Sensu repository
```

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh |
sudo bash


# Install the sensu-go-backend package
sudo apt-get install sensu-go-backend
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh |
sudo bash


# Install the sensu-go-backend package
sudo yum install sensu-go-backend
```

2. Configure and start

You can configure the Sensu backend with `sensu-backend start` flags (recommended) or an `/etc/sensu/backend.yml` file. The Sensu backend requires the `state-dir` flag at minimum, but other useful configurations and templates are available.

> **NOTE**: If you are using Docker, intitialization is included in this step when you start the backend rather than in 3. Initialize. For details about intialization in Docker, read the _backend reference_.

DOCKER

```
# Replace `<username>` and `<password>` with the username and password
# you want to use for your admin user credentials.
docker run -v /var/lib/sensu:/var/lib/sensu \
-d --name sensu-backend \
-p 3000:3000 -p 8080:8080 -p 8081:8081 \
-e SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username> \
-e SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password> \
sensu/sensu:latest \
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

DOCKER

```
# Replace `<username>` and `<password>` with the username and password
```

```yaml
# you want to use for your admin user credentials.
---
version: "3"
services:
  sensu-backend:
    ports:
    - 3000:3000
    - 8080:8080
    - 8081:8081
    volumes:
    - "sensu-backend-data:/var/lib/sensu/sensu-backend/etcd"
    command: "sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-
level debug"
    environment:
    - SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
    - SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
    image: sensu/sensu:latest

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

SHELL

```shell
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/backend.yml -o
/etc/sensu/backend.yml

# Start sensu-backend using a service manager
sudo service sensu-backend start

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

SHELL

```shell
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/backend.yml -o
/etc/sensu/backend.yml

# Start sensu-backend using a service manager
```

```
sudo service sensu-backend start


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

The backend reference includes a complete list of <u>configuration options</u> and <u>backend initialization</u> <u>details</u>

> **WARNING**: If you plan to <u>run a Sensu cluster</u>, make sure that each of your backend nodes is configured, running, and a member of the cluster before you continue the installation process.

3. Initialize

> **NOTE**: If you are using Docker, you already completed intitialization in <u>2. Configure and start</u>.Skip ahead to <u>4. Open the web UI</u> to continue the backend installation process.If you did not use environment variables to override the default admin credentials in step 2, skip ahead to <u>Install sensuctl</u> so you can change your default admin password immediately.

With the backend running, run `sensu-backend init` to set up your Sensu administrator username and password.In this initialization step, you only need to set environment variables with a username and password string — no need for role-based access control (RBAC).

Replace `<username>` and `<password>` with the username and password you want to use.

SHELL

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
sensu-backend init
```

SHELL

```
export SENSU_BACKEND_CLUSTER_ADMIN_USERNAME=<username>
export SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD=<password>
sensu-backend init
```

For details about initializing the Sensu backend, read the backend reference.

4. Open the web UI

The web UI provides a unified view of your observability events and user-friendly tools to reduce alert fatigue. After starting the Sensu backend, open the web UI by visiting http://localhost:3000. You may need to replace `localhost` with the hostname or IP address where the Sensu backend is running.

To log in to the web UI, enter your Sensu user credentials. If you are using Docker and you did not specify environment variables to override the default admin credentials, your user credentials are username `admin` and password `P@ssw0rd!`. Otherwise, your user credentials are the username and password you provided with the `SENSU_BACKEND_CLUSTER_ADMIN_USERNAME` and `SENSU_BACKEND_CLUSTER_ADMIN_PASSWORD` environment variables.

Select the ≡ icon to explore the web UI.

5. Make a request to the health API

To make sure the backend is up and running, use the Sensu health API to check the backend's health. You should receive a response that includes `"Healthy": true`.

```
curl http://127.0.0.1:8080/health
```

Now that you've installed the Sensu backend, install and configure sensuctl to connect to your backend URL. Then you can install a Sensu agent and start monitoring your infrastructure.

Install sensuctl

Sensuctl is a command line tool for managing resources within Sensu. It works by calling Sensu's HTTP API to create, read, update, and delete resources, events, and entities. Sensuctl is available for Linux, Windows, and macOS.

To install sensuctl:

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh |
sudo bash

# Install the sensu-go-cli package
sudo apt-get install sensu-go-cli
```

SHELL

```
# Add the Sensu repository
curl https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh | sudo
bash

# Install the sensu-go-cli package
sudo yum install sensu-go-cli
```

POWERSHELL

```
# Download sensuctl for Windows amd64
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_windows_amd64.zip  -OutFile C:\Users\Administrator\sensu-
go_6.1.4_windows_amd64.zip

# Or for Windows 386
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_windows_386.zip  -OutFile C:\Users\Administrator\sensu-
go_6.1.4_windows_386.zip

# Unzip the file with PowerShell for Windows amd64
Expand-Archive -LiteralPath 'C:\Users\Administrator\sensu-
go_6.1.4_windows_amd64.zip' -DestinationPath 'C:\\Program Files\sensu\sensuctl\bin'

# or for Windows 386
Expand-Archive -LiteralPath 'C:\Users\Administrator\sensu-go_6.1.4_windows_386.zip'
-DestinationPath 'C:\\Program Files\sensu\sensuctl\bin'
```

SHELL

```
# Download the latest release
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go_6.1.4_darwin_amd64.tar.gz

# Extract the archive
tar -xvf sensu-go_6.1.4_darwin_amd64.tar.gz

# Copy the executable into your PATH
sudo cp sensuctl /usr/local/bin/
```

To start using sensuctl, run `sensuctl configure` and log in with your user credentials, namespace, and Sensu backend URL.To configure sensuctl using default values:

```
sensuctl configure -n \
--username 'YOUR_USERNAME' \
--password 'YOUR_PASSWORD' \
--namespace default \
--url 'http://127.0.0.1:8080'
```

Here, the `-n` flag triggers non-interactive mode.Run `sensuctl config view` to view your user profile

For more information about sensuctl, read the sensuctl documentation.

Change default admin password

If you are using Docker and you did not use environment variables to override the default admin credentials in step 2 of the backend installation process, we recommend that you change the default admin password as soon as you have installed sensuctl.Run:

```
sensuctl user change-password --interactive
```

Install Sensu agents

The Sensu agent is available for Ubuntu/Debian, RHEL/CentOS, Windows, and Docker.Review [supported platforms](#) for more information.

1. Download

DOCKER

```
# All Sensu images contain a Sensu backend and a Sensu agent

# Pull the Alpine-based image
docker pull sensu/sensu

# Pull the RHEL-based image
docker pull sensu/sensu-rhel
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh | sudo bash

# Install the sensu-go-agent package
sudo apt-get install sensu-go-agent
```

SHELL

```
# Add the Sensu repository
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh | sudo bash

# Install the sensu-go-agent package
sudo yum install sensu-go-agent
```

POWERSHELL

```
# Download the Sensu agent for Windows amd64
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-go-agent_6.1.4.3866_en-US.x64.msi  -OutFile "$env:userprofile\sensu-go-agent_6.1.4.3866_en-US.x64.msi"

# Or for Windows 386
```

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/6.1.4/sensu-
go-agent_6.1.4.3866_en-US.x86.msi  -OutFile "$env:userprofile\sensu-go-
agent_6.1.4.3866_en-US.x86.msi"


# Install the Sensu agent for Windows amd64
msiexec.exe /i $env:userprofile\sensu-go-agent_6.1.4.3866_en-US.x64.msi /qn


# Or for Windows 386
msiexec.exe /i $env:userprofile\sensu-go-agent_6.1.4.3866_en-US.x86.msi /qn


# Or via Chocolatey
choco install sensu-agent
```

2. Configure and start

You can configure the Sensu agent with `sensu-agent start` flags (recommended) or an
`/etc/sensu/agent.yml` file. The Sensu agent requires the `--backend-url` flag at minimum, but
other useful configurations and templates are available.

DOCKER

```
# If you are running the agent locally on the same system as the Sensu backend,
# add `--link sensu-backend` to your Docker arguments and change the backend
# URL to `--backend-url ws://sensu-backend:8081`.


# Start an agent with the system subscription
docker run -v /var/lib/sensu:/var/lib/sensu -d \
--name sensu-agent sensu/sensu:latest \
sensu-agent start --backend-url ws://sensu.yourdomain.com:8081 --log-level debug --
subscriptions system --api-host 0.0.0.0 --cache-dir /var/lib/sensu
```

DOCKER

```
# Start an agent with the system subscription
---
version: "3"
services:
  sensu-agent:
    image: sensu/sensu:latest
    ports:
```

```
      - 3031:3031
    volumes:
    - "sensu-agent-data:/var/lib/sensu"
    command: "sensu-agent start --backend-url ws://sensu-backend:8081 --log-level
debug --subscriptions system --api-host 0.0.0.0 --cache-dir /var/lib/sensu"

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

SHELL

```
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/agent.yml -o
/etc/sensu/agent.yml

# Start sensu-agent using a service manager
service sensu-agent start
```

SHELL

```
# Copy the config template from the docs
sudo curl -L https://docs.sensu.io/sensu-go/latest/files/agent.yml -o
/etc/sensu/agent.yml

# Start sensu-agent using a service manager
service sensu-agent start
```

POWERSHELL

```
# Copy the example agent config file from
%ALLUSERSPROFILE%\sensu\config\agent.yml.example
# (default: C:\ProgramData\sensu\config\agent.yml.example) to
C:\ProgramData\sensu\config\agent.yml
cp C:\ProgramData\sensu\config\agent.yml.example C:\ProgramData\sensu\config\agent.yml

# Change to the sensu\sensu-agent\bin directory where you installed Sensu
cd 'C:\Program Files\sensu\sensu-agent\bin'

# Run the sensu-agent executable
```

```
./sensu-agent.exe

# Install and start the agent
./sensu-agent service install
```

The agent reference includes a complete list of underline{configuration options}.

3. Verify keepalive events

Sensu keepalives are the heartbeat mechanism used to ensure that all registered agents are operating and can reach the Sensu backend. To confirm that the agent is registered with Sensu and is sending keepalive events, open the entity page in the Sensu web UI or run `sensuctl entity list`.

4. Verify an example event

With your backend and agent still running, send this request to the Sensu events API:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "check-mysql-status"
    },
    "status": 1,
    "output": "could not connect to mysql"
  }
}' \
http://127.0.0.1:3031/events
```

This request creates a `warning` event that you can view in your web UI Events page.

To create an `OK` event, change the `status` to `0` and resend. You can change the `output` value to `connected to mysql` to use a different message for the `OK` event.

Next steps

Now that you have installed Sensu, you're ready to build your observability pipelines! Here are some ideas for next steps.

Get started with Sensu

If you're ready to try Sensu, one of these pathways can get you started:

- Manually trigger an event that sends alerts to your email inbox.

- Create a check to monitor CPU usage and send Slack alerts based on your check.

- Collect metrics with a Sensu check and use a handler to populate metrics in InfluxDB.

- Use the sensuctl dump command to export all of your events and resources as a backup — then use sensuctl create to restore if needed.

Deploy Sensu outside your local development environment

To deploy Sensu for use outside of a local development environment, first decide whether you want to run a Sensu cluster. A Sensu cluster is a group of three or more sensu-backend nodes, each connected to a shared database provided either by Sensu's embedded etcd or an external etcd cluster.

Clustering allows you to absorb the loss of a backend node, prevent data loss, and distribute the network load of agents. However, scaling a single backend to a cluster or migrating a cluster from cleartext HTTP to encrypted HTTPS without downtime can require a number of tedious steps. For this reason, we recommend that you decide whether your deployment will require clustering as part of your initial planning effort.

No matter whether you deploy a single backend or a clustered configuration, begin by securing Sensu with transport layer security (TLS). The first step in setting up TLS is to generate the certificates you need. Then, follow our Secure Sensu guide to make Sensu production-ready.

After you've secured Sensu, read Run a Sensu cluster if you are setting up a clustered configuration.

Commercial features

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

All commercial features are _free for your first 100 entities_.To learn more about Sensu Go commercial licenses for more than 100 entities, _contact the Sensu sales team_.

If you already have a Sensu commercial license, _log in to your Sensu account_ and download your license file.Save your license to a file such as `sensu_license.yml` or `sensu_license.json`.

Use sensuctl to activate your license:

SHELL

```
sensuctl create --file sensu_license.yml
```

SHELL

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

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

```
sensuctl license info
```

Deployment architecture for Sensu

This guide describes various planning considerations and recommendations for a production-ready Sensu deployment, including details related to communication security and common deployment architectures.

etcd is a key-value store that is used by applications of varying complexity, from simple web apps to Kubernetes. The Sensu backend uses an embedded etcd instance for storing both configuration and observability event data, so you can get Sensu up and running without external dependencies.

By building atop etcd, Sensu's backend inherits a number of characteristics to consider when you're planning for a Sensu deployment.

Create and maintain clusters

Sensu's embedded etcd supports initial cluster creation via a static list of peer URLs. After you create a cluster, you can add and remove cluster members with etcdctl tooling.

If you have a healthy clustered backend, you only need to make Sensu API calls to any one of the cluster members. The cluster protocol will replicate your changes to all cluster members.

Read Run a Sensu cluster and the etcd documentation for more information.

Hardware sizing

Because etcd's design prioritizes consistency across a cluster, the speed with which write operations can be completed is very important to the Sensu cluster's performance and health. This means that you should provision Sensu backend infrastructure to provide sustained IO operations per second (IOPS) appropriate for the rate of observability events the system will be required to process.

To maximize Sensu Go performance, we recommend that you:

- Follow our recommended backend hardware configuration.

- Implement documented etcd system tuning practices.

-

- ¬ Benchmark your etcd storage volume to establish baseline IOPS for your system.

- ¬ Scale event storage using PostgreSQL to reduce the overall volume of etcd transactions.

Communications security

Whether you're using using a single Sensu backend or multiple Sensu backends in a cluster, communication with the backend's various network ports (web UI, HTTP API, WebSocket API, etcd client and peer) occurs in cleartext by default. We recommend that you encrypt network communications via TLS, which requires planning and explicit configuration.

Plan TLS for etcd

The URLs for each member of an etcd cluster are persisted to the database after initialization. As a result, moving a cluster from cleartext to encrypted communications requires resetting the cluster, which destroys all configuration and event data in the database. Therefore, we recommend planning for encryption before initiating a clustered Sensu backend deployment.

> **WARNING**: *Reconfiguring a Sensu cluster for TLS post-deployment will require resetting all etcd cluster members, resulting in the loss of all data.*

As described in Secure Sensu, the backend uses a shared certificate and key for web UI and agent communications. You can secure communications with etcd using the same certificate and key. The certificate's common name or subject alternate names must include the network interfaces and DNS names that will point to those systems.

Read Run a Sensu cluster and the etcd documentation for more information about TLS setup and configuration, including a walkthrough for generating TLS certificates for your cluster.

Common Sensu architectures

Depending on your infrastructure and the type of environments you'll be monitoring, you may use one or a combination of these architectures to best fit your needs.

Single backend (standalone)

The single backend (standalone) with embedded etcd architecture requires minimal resources but provides no redundancy in the event of failure.
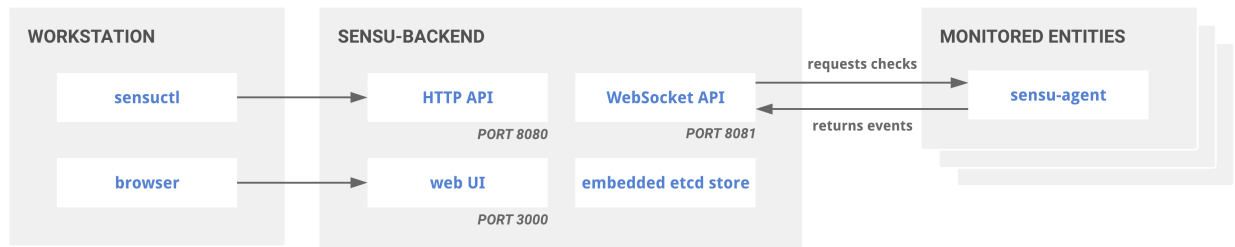
Single Sensu Go backend or standalone architecture

You can reconfigure a single backend as a member of a cluster, but this operation 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.

Clustered deployment for single availability zone

To increase availability and replicate both configuration and data, join the embedded etcd databases of multiple Sensu backend instances together in a cluster.Read Run a Sensu cluster for more information.
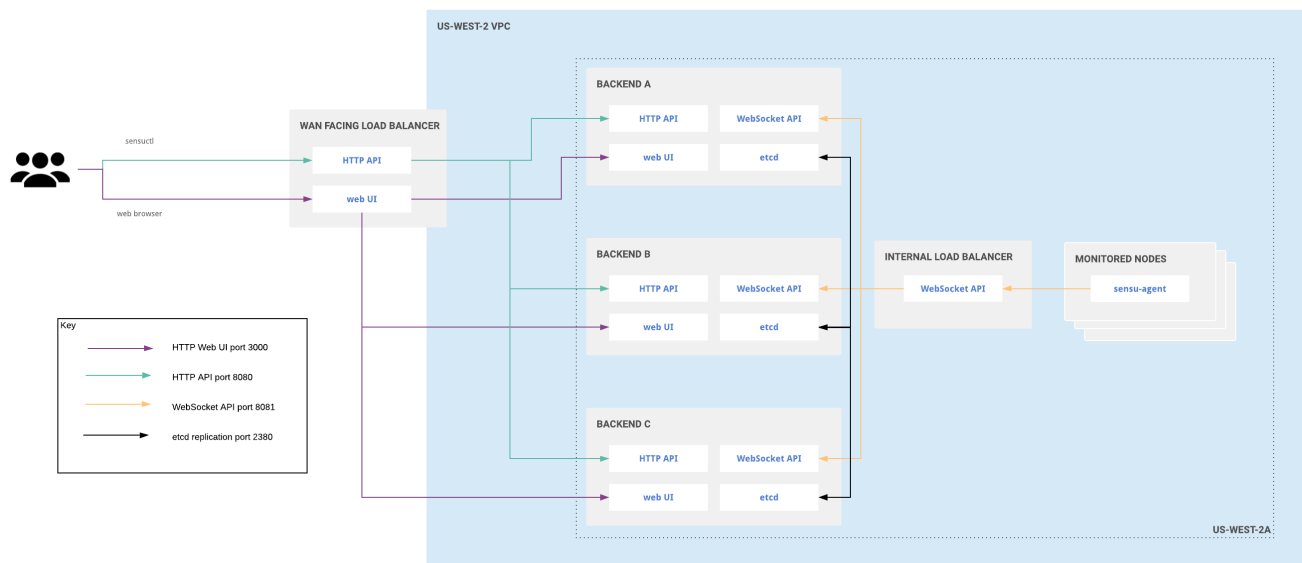
Clustered Sensu Go architecture for a single availability zone

Clustering requires an odd number of backend instances. Although larger clusters provide better fault tolerance, write performance suffers because data must be replicated across more machines. The etcd maintainers recommend clusters of 3, 5, or 7 backends. Read the etcd documentation for more information.

Clustered deployment for multiple availability zones

Distributing infrastructure across multiple availability zones in a given region helps ensure continuous availability of customer infrastructure in the region if any one availability zone becomes unavailable. With this in mind, you can deploy a Sensu cluster across multiple availability zones in a given region, configured to tolerate reasonable latency between those availability zones.
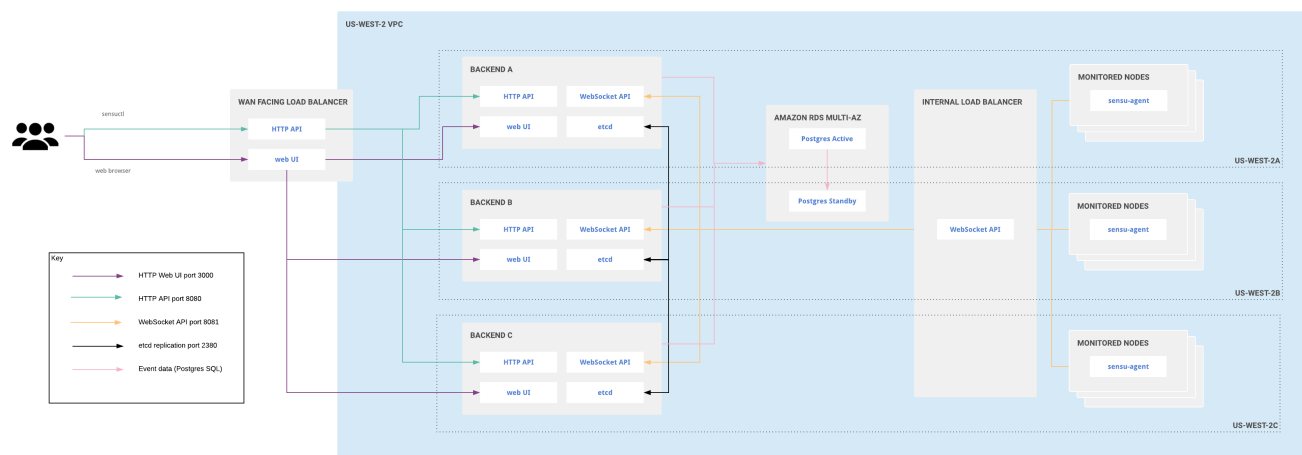
Clustered Sensu Go architecture for multiple availability zones

Large-scale clustered deployment for multiple availability zones

In a large-scale clustered Sensu Go deployment, you can use as many backends as you wish. Use one etcd node per availiability zone, with a minimum of three etcd nodes and a maximum of five. Three etcd nodes allow you to tolerate the loss of a single node with minimal effect on performance. Five etcd nodes allow you to tolerate the loss of two nodes, but with a greater effect on performance.
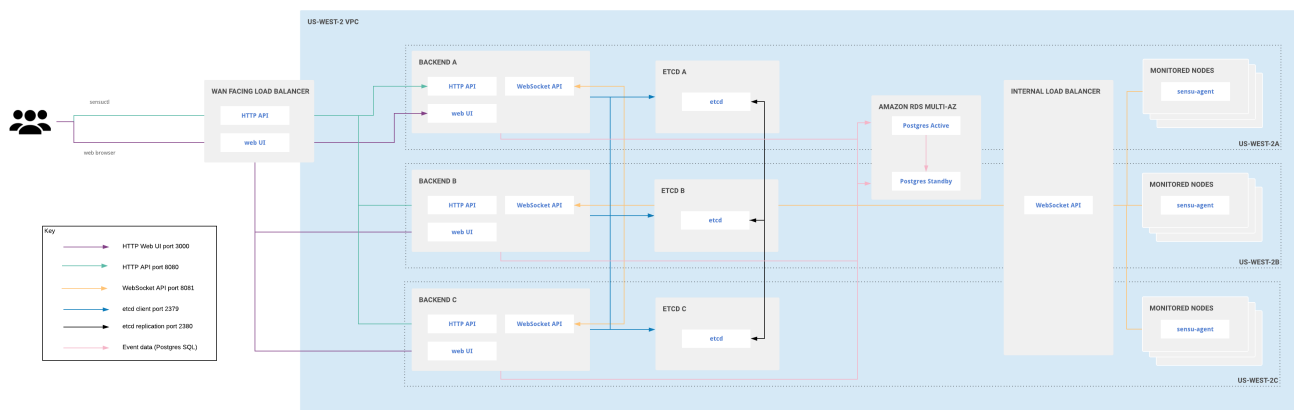
Large-scale clustered Sensu Go architecture for multiple availability zones

Scaled cluster performance with PostgreSQL

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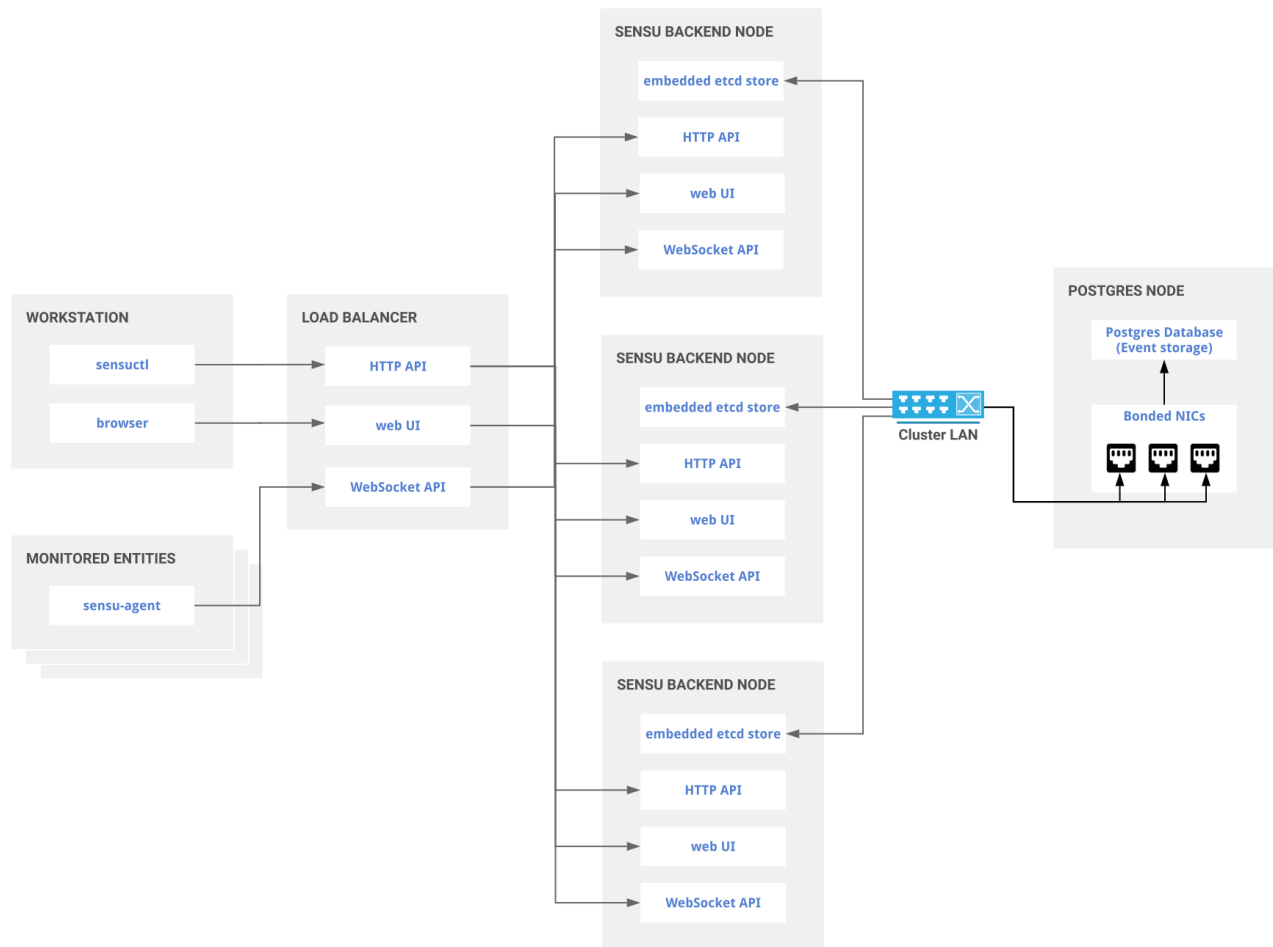
Clustered Sensu Go architecture with PostgreSQL event storage

In load testing, Sensu Go has proven capable of processing 36,000 events per second when using PostgreSQL as the event store.Review 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.

Configuration management

We recommend using configuration management tools to deploy Sensu in production and at scale.

- ¬ Pin versions of Sensu-related software to ensure repeatable Sensu deployments.

- ¬ Ensure consistent configuration between Sensu backends.

The configuration management tools listed here have well-defined Sensu modules to help you get started.

Ansible

The Ansible role to deploy and manage Sensu Go is available in the Sensu Go Ansible Collection.

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

Chef

The Chef cookbook for installing and configuring Sensu is available in the Sensu Go Chef Cookbook.

Contact us for more information about Sensu + Chef.

Puppet

The Puppet module to install Sensu is available in the Sensu Puppet Module.

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

Generate certificates for your Sensu installation

This guide explains how to generate the certificates you need to secure a Sensu cluster and its agents.

When deploying Sensu for use outside of a local development environment, you should secure it using transport layer security (TLS).

TLS uses encryption to provide security for communication between Sensu backends and agents as well as communication between human operators and the Sensu backend, such as web UI or sensuctl access.

Because reconfiguring an existing Sensu deployment from cleartext to TLS can be time-consuming, we recommend that you configure TLS for your backend from the very beginning.

TLS is also required to use some of Sensu's commercial features, like secrets management and mutual TLS authentication (mTLS).

Prerequisites

To use this guide, you must have already installed Sensu on:

- One backend system or three backend systems that you plan to cluster together.

- One or more agents.

Public key infrastructure (PKI)

To use TLS, you must either posses existing public key infrastructure (PKI) or generate your own Certificate Authority (CA) for issuing certificates.

This guide describes how to set up a minimal CA and generate the certificates you need to secure Sensu communications for a clustered backend and agents.

If your organization has existing PKI for certificate issuance, you can adapt the suggestions in this

guide to your organization's PKI.

Recommended practices for deploying and maintaining production PKI can be complex and case-specific, so they are not included in the scope of this guide.

Issue certificates

Use a CA certificate and key to generate certificates and keys to use with Sensu backends and agents.

This example uses the CloudFlare cfssl toolkit to generate a CA and self-signed certificates from that CA.

Install TLS

The cfssl toolkit is released as a collection of command-line tools.

These tools only need to be installed on one system to generate your CA and issue certificates.

You may install the toolkit on your laptop or workstation and store the files there for safekeeping or install the toolkit on one of the systems where you'll run the Sensu backend.

In this example you'll walk through installing cfssl on a Linux system, which requires copying certain certificates and keys to each of the backend and agent systems you are securing.

This guide assumes that you'll install these certificates in the `/etc/sensu/tls` directory on each backend and agent system.

1. Download the cfssl executable:

    ```
    sudo curl -L
    https://github.com/cloudflare/cfssl/releases/download/v1.4.1/cfssl_1.4.1_linux_
    amd64 -o /usr/local/bin/cfssl
    ```

2. Download the cfssljson executable:

    ```
    sudo curl -L
    ```

```
https://github.com/cloudflare/cfssl/releases/download/v1.4.1/cfssljson_1.4.1_li
nux_amd64 -o /usr/local/bin/cfssljson
```

3. Install the cfssl and cfssljson executables in /usr/local/bin::

```
sudo chmod +x /usr/local/bin/cfssl*
```

4. Verify the cfssl executable is version 1.4.1 and runtime go1.12.12:

```
cfssl version
```

5. Verify the cfssljson executable is version 1.4.1 and runtime go1.12.12:

```
cfssljson -version
```

Create a Certificate Authority (CA)

Follow these steps to create a CA with cfssl and cfssljson for each backend and agent system:

1. Create /etc/sensu/tls (which does not exist by default):

```
mkdir -p /etc/sensu/tls
```

2. Navigate to the new /etc/sensu/tls directory:

```
cd /etc/sensu/tls
```

3. Create the CA:

```
echo '{"CN":"Sensu Test CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert
-initca - | cfssljson -bare ca -
```

4. Define signing parameters and profiles (the agent profile provides the "client auth" usage required for mTLS):

```
echo '{"signing":{"default":{"expiry":"17520h","usages":["signing","key
encipherment","client auth"]},"profiles":{"backend":{"usages":["signing","key
encipherment","server auth","client auth"],"expiry":"4320h"},"agent":
{"usages":["signing","key encipherment","client auth"],"expiry":"4320h"}}}}' >
ca-config.json
```

You should now have a directory for each backend and agent at `/etc/sensu/tls` that contains the following files:

filename	description
`ca.pem`	CA root certificate. Must be copied to all systems running Sensu backend or agent.
`ca-key.pem`	CA root certificate private key.
`ca-config.json`	CA signing parameters and profiles. Not used by Sensu.
`ca.csr`	Certificate signing request for the CA root certificate. Not used by Sensu.

The Sensu agent and Sensu backend use the CA root certificate to validate server certificates at connection time.

Generate backend cluster certificates

Now that you've generated a CA, you will use it to generate certificates and keys for each backend server (etcd peer).

For each backend server you'll need to document the IP addresses and hostnames to use in backend and agent communications.

During initial configuration of a cluster of Sensu backends, you must describe every member of the

cluster with a URL passed as the value of the `etcd-initial-cluster` parameter.

In issuing certificates for cluster members, the IP address or hostname used in these URLs must be represented in either the Common Name (CN) or Subject Alternative Name (SAN) records in the certificate.

This guide assumes a scenario with three backend members that are reachable via a `10.0.0.x` IP address, a fully qualified name (for example, `backend-1.example.com`), and an unqualified name (for example, `backend-1`):

Unqualified name	IP address	Fully qualified domain name (FQDN)	Additional names
backend-1	10.0.0.1	backend-1.example.com	localhost, 127.0.0.1
backend-2	10.0.0.2	backend-2.example.com	localhost, 127.0.0.1
backend-3	10.0.0.3	backend-3.example.com	localhost, 127.0.0.1

Note that the additional names for localhost and 127.0.0.1 are added here for convenience and are not strictly required.

Use these name and address details to create two `*.pem` files and one `*.csr` file for each backend.

- ¬ The values provided for the ADDRESS variable will be used to populate the certificate's SAN records.For systems with multiple hostnames and IP addresses, add each to the comma-delimited value of the ADDRESS variable.

- ¬ The value provided for the NAME variable will be used to populate the certificate's CN record.

backend-1

```
export ADDRESS=localhost,127.0.0.1,10.0.0.1,backend-1
export NAME=backend-1.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="backend" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

backend-2

```
export ADDRESS=localhost,127.0.0.1,10.0.0.2,backend-2
export NAME=backend-2.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="backend" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

backend-3

```
export ADDRESS=localhost,127.0.0.1,10.0.0.3,backend-3
export NAME=backend-3.example.com
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl
gencert -config=ca-config.json -profile="backend" -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

You should now have this set of files for each backend:

filename	description	required on backend?
`backend-*.pem`	Backend server certificate	✅
`backend-*-key.pem`	Backend server private key	✅
`backend-*.csr`	Certificate signing request	

Make sure to copy all backend PEM files to the corresponding backend system. For example, the directory listing of /etc/sensu/tls on backend-1 should include:

```
/etc/sensu/tls/
├── backend-1-key.pem
├── backend-1.pem
├── ca.pem
└── ca-key.pem
```

To make sure these files are accessible only by the `sensu` user, run:

```
chown sensu /etc/sensu/tls/*.pem
```

And:

```
chmod 400 /etc/sensu/tls/*.pem
```

Generate agent certificate

Now you will generate a certificate that agents can use to connect to the Sensu backend.

Sensu's commercial distribution offers support for authenticating agents via TLS certificates instead of a username and password.

For this certificate, you only need to specify a CN (here, `agent`) — you don't need to specify an address. You will create the files `agent.pem`, `agent-key.pem`, and `agent.csr`:

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

You should now have a set of files for use by Sensu agents:

filename	description	required on agent?
`agent.pem`	Agent certificate	✅
`agent-key.pem`	Agent private key	✅
`agent.csr`	Certificate signing request	

Make sure to copy all agent PEM files to all agent systems. To continue the example, the directory listing of /etc/sensu/tls on each agent should now include:

```
/etc/sensu/tls/
├── agent-key.pem
├── agent.pem
├── ca-key.pem
└── ca.pem
```

To make sure these files are accessible only by the `sensu` user, run:

```
chown sensu /etc/sensu/tls/*.pem
```

And:

```
chmod 400 /etc/sensu/tls/*.pem
```

Install CA certificates

Before you move on, make sure you have copied the certificates and keys to each of the backend and agent systems you are securing:

- Copy the Certificate Authority (CA) root certificate file, `ca.pem`, to each agent and backend.
- Copy all backend PEM files to their corresponding backend systems.
- Copy all agent PEM files to each agent system.

We also recommend installing the CA root certificate in the trust store of both your Sensu systems and those systems used by operators to manage Sensu.

Installing the CA certificate in the trust store for these systems makes it easier to connect via web UI or sensuctl without being prompted to accept certificates signed by your self-generated CA.

SHELL

```
chmod 644 /etc/sensu/tls/ca.pem
chown root /etc/sensu/tls/ca.pem
sudo apt-get install ca-certificates -y
```

```
sudo ln -sfv /etc/sensu/tls/ca.pem /usr/local/share/ca-certificates/sensu-ca.crt
sudo update-ca-certificates
```

SHELL

```
chmod 644 /etc/sensu/tls/ca.pem
chown root /etc/sensu/tls/ca.pem
sudo yum install -y ca-certificates
sudo update-ca-trust force-enable
sudo ln -s /etc/sensu/tls/ca.pem /etc/pki/ca-trust/source/anchors/sensu-ca.pem
sudo update-ca-trust
```

SHELL

```
Import the root CA certificate on the Mac.
Double-click the root CA certificate to open it in Keychain Access.
The root CA certificate appears in login.
Copy the root CA certificate to System to ensure that it is trusted by all users and
local system processes.
Open the root CA certificate, expand Trust, select Use System Defaults, and save your
changes.
Reopen the root CA certificate, expand Trust, select Always Trust, and save your
changes.
Delete the root CA certificate from login.
```

SHELL

```
Press Windows+R to open the Run dialog.
Type "MMC" (without quotation marks) in the Run dialog and press Enter to open the
MMC console.
In the MMC console, expand the Certificates (Local Computer) node and navigate to
Trusted Root Certification Authorities > Certificates.
Right-click the Trusted Root Certification Authorities > Certificates folder and
select All Tasks > Import to open the Certificate Import wizard.
In the Certificate Import wizard, click Next and browse to the location where the
root CA certificate is stored.
Select the root CA certificate file and click Open.
Click Next, click Next, and click Finish.
```

Renew self-generated certificates

To keep your Sensu deployment running smoothly, renew your self-generated certificates before they expire.Depending on how your certificates are configured, one backend certificate may expire before the others or all three backend certificates may expire at the same time.The agent certificate also expires.

This section explains how to find certificate expiration dates, confirm whether certificates have already expired, and renew certificates.

Find certificate expiration dates

Use this check to find certificate expiration dates so you can renew certificates before they expire and avoid observability interruptions.

Before you run the check, replace `<cert-name>.pem` in the command with the name of the certificate you want to check (for example, `backend-1.pem`).

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  name: expired_certs
  namespace: default
spec:
  command: openssl x509 -noout -enddate -in <cert-name>.pem
  subscriptions:
  - system
  publish: true
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "expired_certs"
```

```
    },
    "spec": {
      "command": "openssl x509 -noout -enddate -in <cert-name>.pem",
      "subscriptions": [
        "system"
      ],
      "publish": true
    }
}
```

The check output will be in the format `notAfter=Month Day HH:MM:SS Year Timezone` .For example:

```
notAfter=Jul  3 22:23:50 2021 GMT
```

Add a handler to send the check output as a notification or to a log file.

Identify expired certificates

The following `sensuctl cluster health` response indicates that one backend certificate is expired:

```
Error: GET "/health": Get https://localhost:8080/health?timeout=3: x509: certificate
has expired or is not yet valid
```

The log for the expired backend will be similar to this example:

```
backend-1      | {"component":"etcd","level":"warning","msg":"health check for peer
a95ca1cdb0b1fcc3 could not connect: remote error: tls: bad certificate (prober
\"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-22T20:40:54Z"}
backend-1      | {"component":"etcd","level":"warning","msg":"health check for peer
a95ca1cdb0b1fcc3 could not connect: remote error: tls: bad certificate (prober
\"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-22T20:40:54Z"}
```

If you restart the cluster with one expired backend certificate, the `sensuctl cluster health`
response will include an error:

```
Error: GET "/health": failed to request new refresh token; client returned 'Post
https://localhost:8080/auth/token: EOF'
```

When all three backend certificates are expired, the log will be similar to this example:

```
backend-1      | {"component":"etcd","level":"warning","msg":"health check for peer
a95ca1cdb0b1fcc3 could not connect: x509: certificate has expired or is not yet valid
(prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-
25T17:49:53Z"}
backend-2      | {"component":"etcd","level":"warning","msg":"health check for peer
4cc36e198efb22e8 could not connect: x509: certificate has expired or is not yet valid
(prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-
25T17:49:16Z"}
backend-3      | {"component":"etcd","level":"warning","msg":"health check for peer
8425a7b2d2ee8597 could not connect: x509: certificate has expired or is not yet valid
(prober \"ROUND_TRIPPER_RAFT_MESSAGE\")","pkg":"rafthttp","time":"2021-06-
25T17:49:16Z"}
```

If you restart the cluster with three expired backend certificates, the `sensuctl cluster health` response will include an error:

```
Error: GET "/health": Get https://127.0.0.1:8080/health?timeout=3: EOF
```

The following `sensuctl cluster health` response helps confirm that all three backend certificates are expired, together with the log warning and restart error examples:

```
=== Etcd Cluster ID: 45c04eab9efc0d11
     ID            Name        Error              Healthy
 ─────────────────── ──────────── ──────────────────────────────── ──────────
 a95ca1cdb0b1fcc3  backend-1   context deadline exceeded  false
 8425a7b2d2ee8597  backend-2   context deadline exceeded  false
 4cc36e198efb22e8  backend-3   context deadline exceeded  false
```

An expired agent certificate does not cause any errors or log messages to indicate the expiration. Use the certificate expiration check to find the agent certificate expiration date.

Renew certificates

To renew your certificates, whether they expired or not, follow the steps to create a CA, generate backend certificates, or generate an agent certificate. The new certificate will override the existing certificate.

After you save the new certificates, restart each backend:

```
sudo systemctl start sensu-backend
```

Next step: Secure Sensu

Now that you have generated the required certificates and copied them to the applicable hosts, follow Secure Sensu to make your Sensu installation production-ready.

Secure Sensu

As with any piece of software, it is critical to minimize any attack surface the software exposes. Sensu is no different.

This reference describes the components you need to secure to make Sensu production-ready, including etcd peer communication, the Sensu API and web UI, and Sensu agent-to-server communication. It also describes agent mutual transport layer security (mTLS) authentication, which is required for secrets management.

Before you can secure Sensu, you must generate the certificates you will need. After you generate certificates, follow this reference to secure Sensu for production.

Secure etcd peer communication

> **WARNING**: *You must update the default configuration for Sensu's embedded etcd with an explicit, non-default configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

To properly secure etcd communication, replace the default parameter values in your backend store configuration in `/etc/sensu/backend.yml` as follows:

1. Replace the placeholder with the path to your certificate and key for the `etcd-cert-file` and `etcd-key-file` to secure client communication:

```
etcd-cert-file: "/etc/sensu/tls/backend-1.pem"
etcd-key-file: "/etc/sensu/tls/backend-1-key.pem"
```

2. Replace the placeholder with the path to your certificate and key for the `etcd-peer-cert-file` and `etcd-peer-key-file` to secure cluster communication:

```
etcd-peer-cert-file: "/etc/sensu/tls/backend-1.pem"
```

```
etcd-peer-key-file: "/etc/sensu/tls/backend-1-key.pem"
```

3. Replace the placeholder with the path to your `ca.pem` certificate for the `etcd-trusted-ca-file` and `etcd-peer-trusted-ca-file` to secure communication with the etcd client server and between etcd cluster members:

```
etcd-trusted-ca-file: "/etc/sensu/tls/ca.pem"
etcd-peer-trusted-ca-file: "/etc/sensu/tls/ca.pem"
```

4. Add non-default values for `etcd-listen-client-urls`, `etcd-listen-peer-urls`, and `etcd-initial-advertise-peer-urls`:

```
etcd-listen-client-urls: "https://localhost:2379"
etcd-listen-peer-urls: "https://localhost:2380"
etcd-advertise-client-urls: "https://localhost:2379"
etcd-initial-advertise-peer-urls: "https://localhost:2380"
```

> **NOTE**: If you are securing a <u>cluster</u>, use your backend node IP address instead of `localhost` in the non-default values for `etcd-listen-client-urls`, `etcd-listen-peer-urls`, and `etcd-initial-advertise-peer-urls`.

5. Set `etcd-client-cert-auth` and `etcd-peer-client-cert-auth` to `true` to ensure that etcd only allows connections from clients and peers that present a valid, trusted certificate:

```
etcd-client-cert-auth: "true"
etcd-peer-client-cert-auth: "true"
```

Because etcd does not require authentication by default, you must set the `etcd-client-cert-auth` and `etcd-peer-client-cert-auth` parameters to `true` to secure Sensu's embedded etcd datastore against unauthorized access.

> **NOTE**: The <u>Sensu backend reference</u> includes more information about each etcd store parameter.

Secure the Sensu agent API, HTTP API, and web UI

The Sensu Go agent API, HTTP API, and web UI use a common stanza in `/etc/sensu/backend.yml` to provide the certificate, key, and CA file needed to provide secure communication.

> **NOTE**: *By changing these parameters, the server will communicate using transport layer security (TLS) and expect agents that connect to it to use the WebSocket secure protocol. For communication to continue, you must complete the configuration in this section **and** in the Sensu agent-to-server communication section.*

Configure the following backend secure sockets layer (SSL) attributes in `/etc/sensu/backend.yml`:

1. Replace the placeholders with the paths to your CA root, backend certificate, and backend key files for the `trusted-ca-file`, `cert-file`, and `key-file` parameters:

   ```
   trusted-ca-file: "/etc/sensu/tls/ca.pem"
   cert-file: "/etc/sensu/tls/backend-1.pem"
   key-file: "/etc/sensu/tls/backend-1-key.pem"
   ```

2. Set the `insecure-skip-tls-verify` parameter to `false`:

   ```
   insecure-skip-tls-verify: false
   ```

3. When you provide these cert-file and key-file parameters, the agent WebSocket API and HTTP API will serve requests over SSL/TLS (https). For this reason, you must also specify `https://` schema for the `api-url` parameter for backend API configuration:

   ```
   api-url: "https://localhost:8080"
   ```

After you restart the `sensu-backend` service, the parameters will load and you will able to access the web UI at https://localhost:3000. Configuring these attributes will also ensure that agents can communicate securely.

Specify a separate web UI certificate and key

You can use the same certificates and keys to secure etcd, the HTTP API, and the web UI. However, if you prefer, you can use a separate certificate and key for the web UI (for example, a commercially purchased certificate and key).

To do this, add the `dashboard-cert-file` and `dashboard-key-file` parameters for backend SSL configuration in `/etc/sensu/backend.yml`:

```
dashboard-cert-file: "/etc/sensu/tls/separate-webui-cert.pem"
dashboard-key-file: "/etc/sensu/tls/separate-webui-key.pem"
```

Secure Sensu agent-to-server communication

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

```
backend-url:
  - "ws://127.0.0.1:8081"
```

To use WebSocket over SSL/TLS (wss), change the `backend-url` value to the `wss://` schema in `/etc/sensu/agent.yml` :

```
backend-url:
  - "wss://127.0.0.1:8081"
```

The agent will connect to Sensu backends over wss.Remember, if you change the configuration to wss, plaintext communication will not be possible.

You can also provide a trusted CA root certificate file as part of the agent configuration (named `ca.pem` in the example in Generate certificates).If you will start the agent via `sensu-agent start` , pass the `--trusted-ca-file` flag with the start command.Otherwise, include the `trusted-ca-file` parameter in the agent configuration in `/etc/sensu/agent.yml` :

```
trusted-ca-file: "/etc/sensu/tls/ca.pem"
```

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

Optional: Configure Sensu agent mTLS authentication

> *COMMERCIAL FEATURE: Access client mutual transport layer security (mTLS) authentication in the packaged Sensu Go distribution.For more information, read Get started with commercial features.*

By default, Sensu agents require username and password authentication to communicate with Sensu backends.For Sensu's default user credentials and details about configuring Sensu role-based access control (RBAC), read the RBAC reference.

Alternatively, Sensu agents can use mTLS for authenticating to the backend WebSocket transport. When agent mTLS authentication is enabled, agents do not need to send password credentials to backends when they connect.To use secrets management, Sensu agents must be secured with mTLS. In addition, when using mTLS authentication, agents do not require an explicit user in Sensu.Sensu agents default to authenticating as the `agent` user and using permissions granted to the

`system:agents` group by the `system:agents` cluster role and cluster role binding.

You can still bind agents to a specific user when the `system:agents` group is problematic. For this use case, create a user that matches the Common Name (CN) of the agent's certificate.

> **NOTE**: Sensu agents need to be able to create events in the agent's namespace. To ensure that agents with incorrect CN fields can't access the backend, remove the default `system:agents` group.

To view a certificate's CN with openssl, run:

```
openssl x509 -in client.pem -text -noout
```

The response should be similar to this example:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            37:57:7b:04:1d:67:63:7b:ff:ae:39:19:5b:55:57:80:41:3c:ec:ff
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = CA
        Validity
            Not Before: Sep 26 18:58:00 2019 GMT
            Not After : Sep 24 18:58:00 2024 GMT
        Subject: CN = client
...
```

The `Subject:` field indicates the certificate's CN is `client`, so to bind the agent to a particular user in Sensu, create a user called `client`.

To enable agent mTLS authentication:

1. Create and distribute a new Certificate Authority (CA) root certificate and new agent and backend certificates and keys according to the Generate certificates guide.

2. Add the following parameters and values to the backend configuration:

```
agent-auth-cert-file: "/etc/sensu/tls/backend-1.pem"
agent-auth-key-file: "/etc/sensu/tls/backend-1-key.pem"
agent-auth-trusted-ca-file: "/etc/sensu/tls/ca.pem"
```

3. Add the following parameters and values to the agent configuration:

```
cert-file: "/etc/sensu/tls/agent.pem"
key-file: "/etc/sensu/tls/agent-key.pem"
trusted-ca-file: "/etc/sensu/tls/ca.pem"
```

You can use use certificates for authentication that are distinct from other communication channels used by Sensu, like etcd or the API.However, deployments can also use the same certificates and keys for etcd peer and client communication, the HTTP API, and agent authentication without issues.

Certificate revocation check

The Sensu backend checks certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP) endpoints for agent mTLS, etcd client, and etcd peer connections whose remote sides present X.509 certificates that provide CRL and OCSP revocation information.

Next step: Run a Sensu cluster

Well done!Your Sensu installation should now be secured with TLS.The last step before you deploy Sensu is to set up a Sensu cluster.

Run a Sensu cluster

To deploy Sensu for use outside of a local development environment, first decide whether you want to run a Sensu cluster.

A Sensu cluster is a group of at least three sensu-backend nodes, each connected to a shared database provided either by Sensu's embedded etcd or an external etcd cluster. Creating a Sensu cluster ultimately configures an etcd cluster.

Clustering improves Sensu's availability, reliability, and durability. It allows you to absorb the loss of a backend node, prevent data loss, and distribute the network load of agents. If you have a healthy clustered backend, you only need to make Sensu API calls to any one of the cluster members. The cluster protocol will replicate your changes to all cluster members.

Scaling a single backend to a cluster or migrating a cluster from cleartext HTTP to encrypted HTTPS without downtime can require a number of tedious steps. For this reason, we recommend that you **decide whether your deployment will require clustering as part of your initial planning effort**.

No matter whether you deploy a single backend or a clustered configuration, begin by securing Sensu with transport layer security (TLS). The first step in setting up TLS is to generate the certificates you need. Then, follow our Secure Sensu guide to make Sensu production-ready.

After you've secured Sensu, continue reading this document to set up and update a clustered configuration.

> **NOTE**: *We recommend using a load balancer to evenly distribute agent connections across a cluster.*

Configure a cluster

The sensu-backend arguments for its store mirror the etcd configuration flags, but the Sensu flags are prefixed with `etcd`. For more detailed descriptions of the different arguments, read the etcd documentation or Sensu backend reference.

You can configure a Sensu cluster in a couple different ways — we'll show you a few below — but you should adhere to some etcd cluster guidelines as well:

> The recommended etcd cluster size is 3, 5 or 7, which is decided by the fault tolerance requirement. A 7-member cluster can provide enough fault tolerance in most cases. While a larger cluster provides better fault tolerance, the write performance reduces since data needs to be replicated to more machines. It is recommended to have an odd number of members in a cluster. Having an odd cluster size doesn't change the number needed for majority, but you gain a higher tolerance for failure by adding the extra member. *etcd2 Admin Guide*

We also recommend using stable platforms to support your etcd instances (review etcd's supported platforms).

> **NOTE**: *If a cluster member is started before it is configured to join a cluster, the member will persist its prior configuration to disk. For this reason, you must remove any previously started member's etcd data by stopping sensu-backend and deleting the contents of `/var/lib/sensu/sensu-backend/etcd` before proceeding with cluster configuration.*

Docker

If you prefer to stand up your Sensu cluster within Docker containers, check out the Sensu Go Docker configuration. This configuration defines three sensu-backend containers and three sensu-agent containers.

Traditional computer instance

> **NOTE**: *The remainder of this guide describes on-disk configuration. If you are using an ephemeral computer instance, you can use `sensu-backend start --help` to list etcd command line flags. The configuration file entries in the rest of this guide translate to `sensu-backend` flags.*

Sensu backend configuration

> **WARNING**: *You must update the default configuration for Sensu's embedded etcd with an explicit, non-default configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

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

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: "unique_token_for_this_cluster"
etcd-name: "backend-1"
```

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: "unique_token_for_this_cluster"
etcd-name: "backend-2"
```

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: "unique_token_for_this_cluster"
etcd-name: "backend-3"
```

> **WARNING**: To properly secure etcd communication, replace the default URLs for `etcd-advertise-client-urls`, `etcd-listen-client-urls`, `etcd-listen-peer-urls`, and `etcd-initial-cluster` in the store configurations for your backends with non-default values.
>
> Specify the same `etcd-initial-cluster-token` value for all three backends. This allows etcd to generate unique cluster IDs and member IDs even for clusters that have otherwise identical configurations and prevents cross-cluster-interaction.

After you configure each node as described in these examples, start each sensu-backend:

```
sudo systemctl start sensu-backend
```

Add Sensu agents to clusters

Each Sensu agent should have the following entries in `/etc/sensu/agent.yml` to ensure the agent is aware of all cluster members. This allows the agent to reconnect to a working backend if the backend it is currently connected to goes into an unhealthy state.

Here is an example backend-url configuration for all agents connecting to the cluster over WebSocket:

```
backend-url:
  - "ws://10.0.0.1:8081"
  - "ws://10.0.0.2:8081"
  - "ws://10.0.0.3:8081"
```

You should now have a highly available Sensu cluster! Confirm cluster health and try other cluster management commands with sensuctl.

Manage and monitor clusters with sensuctl

Sensuctl includes several commands to help you manage and monitor your cluster.Run `sensuctl cluster -h` for additional help information.

Get cluster health status

Get cluster health status and etcd alarm information:

```
sensuctl cluster health
```

The cluster health response will list the health status for each cluster member, similar to this example:

```
ID                Name          Error                                        Healthy
_____  _____  _____  _____
_____

a32e8f613b529ad4  backend-1                                                  true
c3d9f4b8d0dd1ac9  backend-2  dial tcp 10.0.0.2:2379: connect: connection refused  false
c8f63ae435a5e6bf  backend-3                                                  true
```

Add a cluster member

To add a new member node to an existing cluster:

1. Configure the new node's store in its `/etc/sensu/backend.yml` configuration file.For the new node `backend-4` with IP address `10.0.0.4`:

   ```
   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: "unique_token_for_this_cluster"
etcd-name: "backend-4"
```

> **NOTE**: To make sure the new member is added to the correct cluster, specify the same `etcd-initial-cluster-token` value that you used for the other members in the cluster.
>
> Also, when you are adding a cluster member, make sure the `etcd-initial-cluster-state` value is `existing`, **not** `new`.

2. Run the sensuctl command to add the new cluster member:

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

You will receive a sensuctl response to confirm that the new member was added:

```
added member 2f7ae42c315f8c2d to cluster
```

3. Start the new backend:

```
sudo systemctl start sensu-backend
```

4. Add the new cluster member's WebSocket backend-url in `/etc/sensu/agent.yml` for all agents that connect to the cluster over WebSocket:

```
backend-url:
  - "ws://10.0.0.1:8081"
  - "ws://10.0.0.2:8081"
  - "ws://10.0.0.3:8081"
  - "ws://10.0.0.4:8081"
```

List cluster members

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

```
sensuctl cluster member-list
```

You will receive a sensuctl response that lists all cluster members:

```
      ID            Name       Peer URLs           Client URLs
_____ _____ _____ _____

a32e8f613b529ad4   backend-1   http://10.0.0.1:2380    http://10.0.0.1:2379
c3d9f4b8d0dd1ac9   backend-2   http://10.0.0.2:2380    http://10.0.0.2:2379
c8f63ae435a5e6bf   backend-3   http://10.0.0.3:2380    http://10.0.0.3:2379
2f7ae42c315f8c2d   backend-4   http://10.0.0.4:2380    http://10.0.0.4:2379
```

Remove a cluster member

Remove a faulty or decommissioned member node from a cluster:

```
sensuctl cluster member-remove 2f7ae42c315f8c2d
```

You will receive a sensuctl response to confirm that the cluster member was removed:

```
Removed member 2f7ae42c315f8c2d from cluster
```

Replace a faulty cluster member

To replace a faulty cluster member to restore a cluster's health:

1. Get cluster health status and etcd alarm information:

   ```
   sensuctl cluster health
   ```

In the response, for a faulty cluster member, the Error column will include an error message and the Healthy column will list `false` .In this example, the response indicates that cluster member `backend-4` is faulty:

```
    ID          Name                Error                    Healthy
 _____ _____
 _____ _____

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

2. Remove the faulty cluster member — in this example, `backend-4` — using its ID.Removing the faulty cluster member prevents the cluster size from growing.

```
sensuctl cluster member-remove 2f7ae42c315f8c2d
```

The response should indicate that the cluster member was removed:

```
Removed member 2f7ae42c315f8c2d from cluster
```

3. Follow the steps in Add a cluster member to configure the replacement cluster member.

> **NOTE**: You can use the same name and IP address as the removed faulty member for the replacement cluster member.When updating the replacement member's backend configuration file, make sure the `etcd-initial-cluster-state` value is `existing` , **not** `new` .

If replacing the faulty cluster member does not resolve the problem, read the etcd operations guide for more information.

Update a cluster member

Update the peer URLs of a member in a cluster:

```
sensuctl cluster member-update c8f63ae435a5e6bf http://10.0.0.4:2380
```

You will receive a sensuctl response to confirm that the cluster member was updated:

```
Updated member with ID c8f63ae435a5e6bf in cluster
```

Cluster security

Read Secure Sensu for information about cluster security.

Use an external etcd cluster

> **WARNING**: *You must update the example configuration for external etcd with an explicit, non-default configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

To use Sensu with an external etcd cluster, you must have etcd 3.3.2 or newer. To stand up an external etcd cluster, follow etcd's clustering guide using the same store configuration. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

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 paths to certificates created using your Certificate Authority (CA) and a list of etcd client URLs:

```
sensu-backend start \
--etcd-trusted-ca-file=./ca.pem \
--etcd-cert-file=./backend-1.pem \
--etcd-key-file=./backend-1-key.pem \
--etcd-client-urls='https://10.0.0.1:2379 https://10.0.0.2:2379
https://10.0.0.3:2379' \
--no-embed-etcd
```

> **NOTE**: The etcd and sensu-backend certificates must share a CA, and the `etcd-client-urls` value must be a space-delimited list or a YAML array.

Migrate from embedded etcd to external etcd

To migrate from embedded etcd to external etcd, first decide whether you need to migrate all of your etcd data or just your Sensu configurations.

If you need to migrate all etcd data, you must create an etcd snapshot. Use the snapshot to restore you entire cluster after setting up the new external cluster.

If you need to migrate only your Sensu configuration, you can use sensuctl dump to create a backup and use sensuctl create to import your configuration to the new external cluster.

> **NOTE**: The sensuctl dump command does not export user passwords, and sensuctl create does not restore API keys from a sensuctl dump backup. For this reason, you must use the etcd snapshot and restore process to migrate your entire embedded cluster to external etcd.

After you create the backups you need, follow Use an external etcd cluster to configure Sensu to use the external cluster as your datastore.

Troubleshoot clusters

Failure modes

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

Disaster recovery

Read the etcd recovery guide for disaster recovery information.

Redeploy a cluster

To redeploy a cluster due to an issue like loss of quorum among cluster members, etcd corruption, or hardware failure, read Remove and redeploy a cluster.

Multi-cluster visibility with federation

Sensu's federation API allows you to register external clusters, gain visibility into the health of your infrastructure and services across multiple distinct Sensu instances within a single web UI, and mirror your changes in one cluster to follower clusters. This is useful when you want to provide a single entry point for Sensu users who need to manage monitoring across multiple distinct physical data centers, cloud regions, or providers.
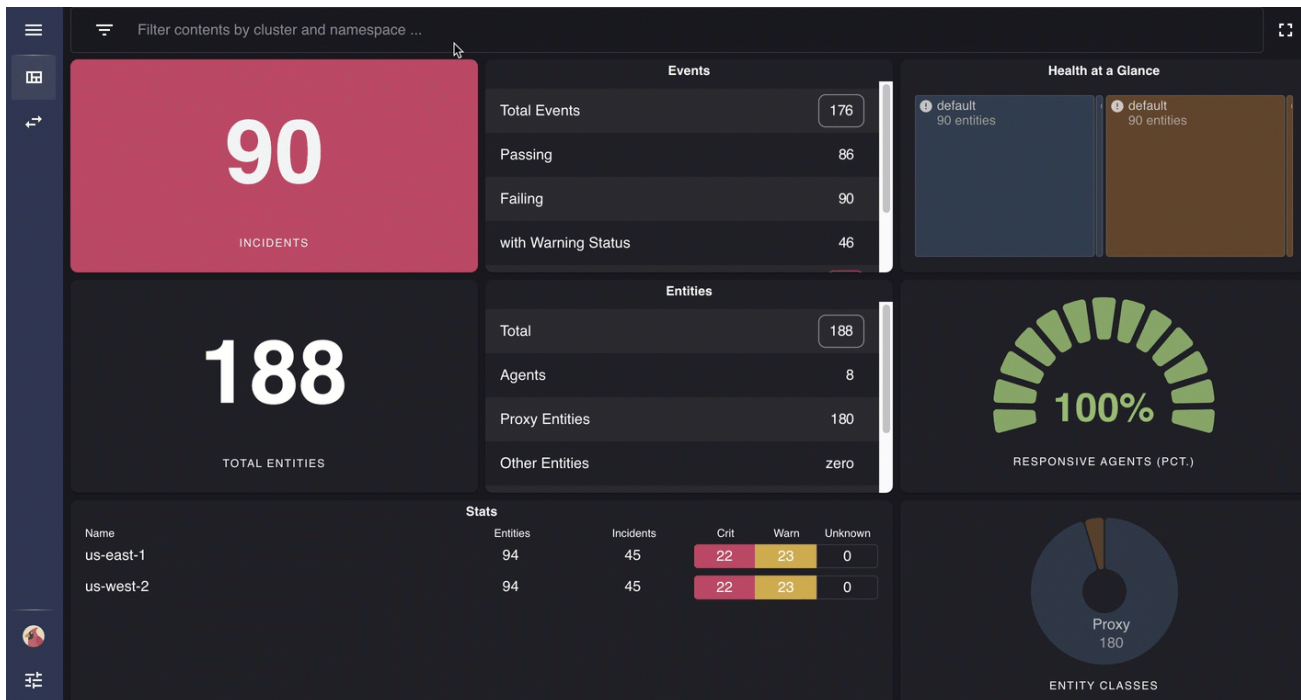
After you configure federation, you can also create, update, and delete clusters using sensuctl create, edit, and delete commands.

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

Only cluster administrators can register a new cluster, but every user can query the list of clusters.

Complete federation of multiple Sensu instances relies on a combination of features:

Feature	Purpose in federation
JSON Web Token (JWT) authentication	Cross-cluster token authentication using asymmetric key encryption
etcd replicators	Replicate RBAC policy across clusters and namespaces
Federation Gateway and APIs	Configure federation access for cross-cluster visibility in web UI

Follow the example in this guide to configure these features. The example assumes you wish to federate three named Sensu clusters:

Cluster name	Hostname
`gateway`	sensu.gateway.example.com
`alpha`	sensu.alpha.example.com
`beta`	sensu.beta.example.com

In this example, the `gateway` cluster will be the entry point for operators to manage Sensu resources in the `alpha` and `beta` clusters. This guide assumes a single sensu-backend in each cluster, but named clusters composed of multiple sensu-backends are supported.

This diagram depicts the federation relationship documented in this guide:

Example federated architecture
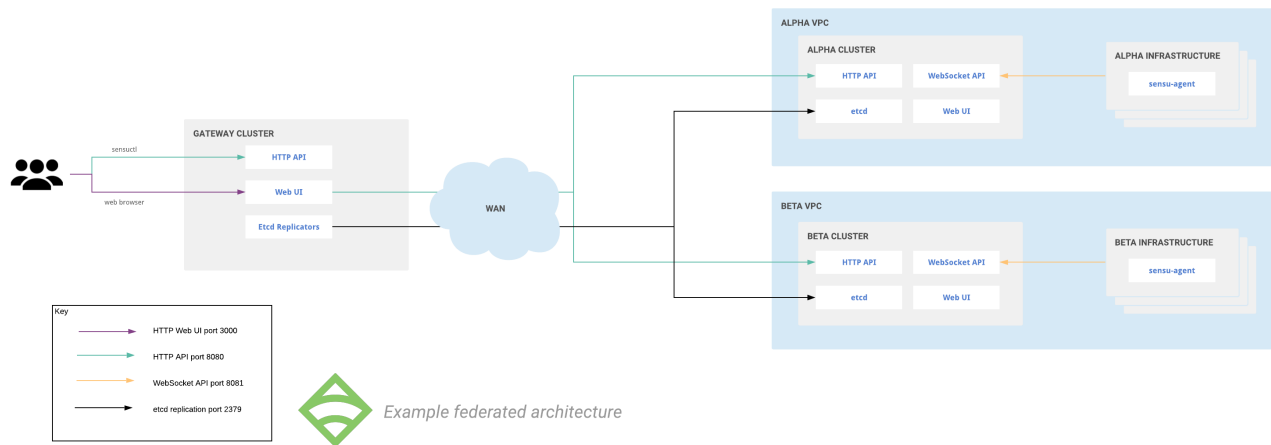
Complete the steps in this guide to browse events, entities, checks, and other resources in the `gateway`, `alpha`, and `beta` clusters from the `gateway` cluster web UI.

Configure backends for TLS

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

To ensure that cluster members can validate each other, certificates for each cluster member should include the IP addresses or hostnames specified in the values of sensu-backend `etcd-advertise-client-urls`, `etcd-advertise-peer-urls`, and `etcd-initial-advertise-peer-urls` parameters. In addition to the certificate's Common Name (CN), Subject Alternative Names (SANs) are also honored for validation.

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

- A PEM-formatted X.509 certificate and corresponding private key copied to each cluster member.
- A corresponding certificate authority (CA) certificate chain copied to each cluster member.

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

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

Backend property	Description
`etcd-cert-file`	Path to certificate used for TLS on etcd client/peer communications (for example, `/etc/sensu/tls/backend-1.pem`.
`etcd-key-file`	Path to key corresponding with `etcd-cert-file` certificate (for example, `/etc/sensu/tls/backend-1-key.pem`.
`etcd-trusted-ca-file`	Path to CA certificate chain file (for example, `/etc/sensu/tls/ca.pem`. This CA certificate chain must be usable to validate certificates for all backends in the federation.
`etcd-client-cert-auth`	Enforces certificate validation to authenticate etcd replicator connections. Set to `true` to secure etcd communication.
`etcd-advertise-client-urls`	List of https URLs to advertise for etcd replicators, accessible by other backends in the federation (for example, `https://sensu.beta.example.com:2379`).

`etcd-listen-client-urls`	List of https URLs to listen on for etcd replicators (for example, `https://0.0.0.0:2379` to listen on port 2379 across all ipv4 interfaces).

> **WARNING**: *You must provide an explicit, non-default etcd configuration to secure etcd communication in transit. If you do not properly configure secure etcd communication, your Sensu configuration will be vulnerable to unauthorized manipulation via etcd client connections.*

This includes providing non-default values for the `etcd-advertise-client-urls` and `etcd-listen-client-urls` backend parameters and creating a [certificate and key](#) for the `etcd-cert-file` and `etcd-key-file` values. The default values are not suitable for use under federation.

Configure shared token signing keys

Whether federated or standalone, Sensu backends issue JSON Web Tokens (JWTs) to users upon successful authentication. These tokens include a payload that describes the username and group affiliations. The payload is used to determine permissions based on the configured [RBAC policy](#).

In a federation of Sensu backends, each backend needs to have the same public/private key pair. These asymmetric keys are used to crypotgraphically vouch for the user's identity in the JWT payload. Using shared JWT keys enables clusters to grant users access to Sensu resources according to their local policies but without requiring user resources to be present uniformly across all clusters in the federation.

By default, a Sensu backend automatically generates an asymmetric key pair for signing JWTs and stores it in the etcd database. When configuring federation, you must generate keys as files on disk so they can be copied to all backends in the federation.

1. Use the `openssl` command line tool to generate a P-256 elliptic curve private key:

   ```
   openssl ecparam -genkey -name prime256v1 -noout -out jwt_private.pem
   ```

2. Generate a public key from the private key:

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

3. Save the JWT keys in `/etc/sensu/certs` on each cluster backend.

4. Add the `jwt-private-key-file` and `jwt-public-key-file` attributes in `/etc/sensu/backend.yml` and specify the paths to the JWT private and public keys:

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

5. Restart the Sensu backend so that your settings take effect:

```
sensu-backend start
```

Add a user and a cluster role binding

To test your configuration, provision a user and a cluster role binding in the `gateway` cluster.

1. Confirm that sensuctl is configured to communicate with the `gateway` cluster:

```
sensuctl config view
```

The response will list the active configuration:

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

2. Create a `federation-viewer` user:

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

3. When prompted for username and groups, press enter.

4. When prompted for password, enter a password for the `federation-viewer` user. Make a note of the password you entered — you'll use it to log in to the web UI after you configure RBAC policy replication and registered clusters into your federation.

   This creates the following user:

   YML

   ```
   username: federation-viewer
   disabled: false
   ```

   JSON

   ```
   {
     "username": "federation-viewer",
     "disabled": false
   }
   ```

5. Grant the `federation-viewer` user read-only access with a cluster role binding for the built-in `view` cluster role:

   ```
   sensuctl cluster-role-binding create federation-viewer-readonly --cluster-
   role=view --user=federation-viewer
   ```

   This command creates the following cluster role binding resource definition:

   YML

   ```
   type: ClusterRoleBinding
   api_version: core/v2
   metadata:
     created_by: admin
     name: federation-viewer-readonly
   spec:
   ```

```yaml
    role_ref:
      name: view
      type: ClusterRole
    subjects:
    - name: federation-viewer
      type: User
```

JSON

```json
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "federation-viewer-readonly"
  },
  "spec": {
    "role_ref": {
      "name": "view",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "federation-viewer",
        "type": "User"
      }
    ]
  }
}
```

Create etcd replicators

Etcd replicators use the etcd make-mirror utility for one-way replication of Sensu RBAC policy resources. This allows you to centrally define RBAC policy on the `gateway` cluster and replicate RBAC resources to other clusters in the federation (`alpha` and `beta`), ensuring consistent permissions for Sensu users across multiple clusters via the `gateway` web UI.

1. Configure one etcd replicator per cluster for each RBAC policy resource, across all namespaces, for each backend in the federation.

In this example, the following etcd replicator resources will replicate ClusterRoleBinding resources from the `gateway` cluster to the two target clusters:

YML

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

JSON

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

```
      "resource": "ClusterRoleBinding",
      "replication_interval_seconds": 30
    }
  }
```

YML

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

JSON

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

2. Run `sensuctl config view` and verify that `sensuctl` is configured to talk to a `gateway` cluster API.Reconfigure `sensuctl` if needed.

3. Save the `AlphaClusterRoleBindings` and `BetaClusterRoleBindings` EtcdReplicator definitions to a file (for example, `etcdreplicators.yml` or `etcdreplicators.json`).

4. Use `sensuctl create -f` to apply the `AlphaClusterRoleBindings` and `BetaClusterRoleBindings` EtcdReplicator definitions to the `gateway` cluster:

   SHELL

   ```
   sensuctl create -f etcdreplicators.yml
   ```

   SHELL

   ```
   sensuctl create -f etcdreplicators.json
   ```

5. Verify that the EtcdReplicator resource is working as expected: reconfigure the sensuctl backend URL to communicate with the `alpha` and `beta` clusters and run the following command for each:

   ```
   sensuctl cluster-role-binding info federation-viewer-readonly
   ```

   The `federation-viewer-readonly` binding you created in the previous section should be listed in the output from each cluster:

   ```
   === federation-viewer-readonly
   Name:         federation-viewer-readonly
   Cluster Role: view
   Subjects:
     Users:      federation-viewer
   ```

Register clusters

Clusters must be registered to become visible in the web UI.Each registered cluster must have a name and a list of one or more cluster member URLs corresponding to the backend REST API.

> **NOTE**: Individual cluster resources may list the API URLs for a single stand-alone backend or multiple backends that are members of the same etcd cluster.Creating a cluster resource that lists multiple backends that do not belong to the same cluster will result in unexpected behavior.

Register a single cluster

With `sensuctl` configured for the `gateway` cluster, run `sensuctl create` on the yaml or JSON below to register cluster `alpha`:

YML

```yml
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

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

JSON

```json
{
  "api_version": "federation/v1",
  "type": "Cluster",
  "metadata": {
    "name": "beta"
  },
  "spec": {
    "api_urls": [
      "https://sensu.alpha.example.com:8080"
    ]
  }
}
```

> **NOTE**: *When logging into the* `gateway` *cluster web UI, any namespaces, entities, events, and other resources specific to that cluster will be labeled as* `local-cluster` *.*

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

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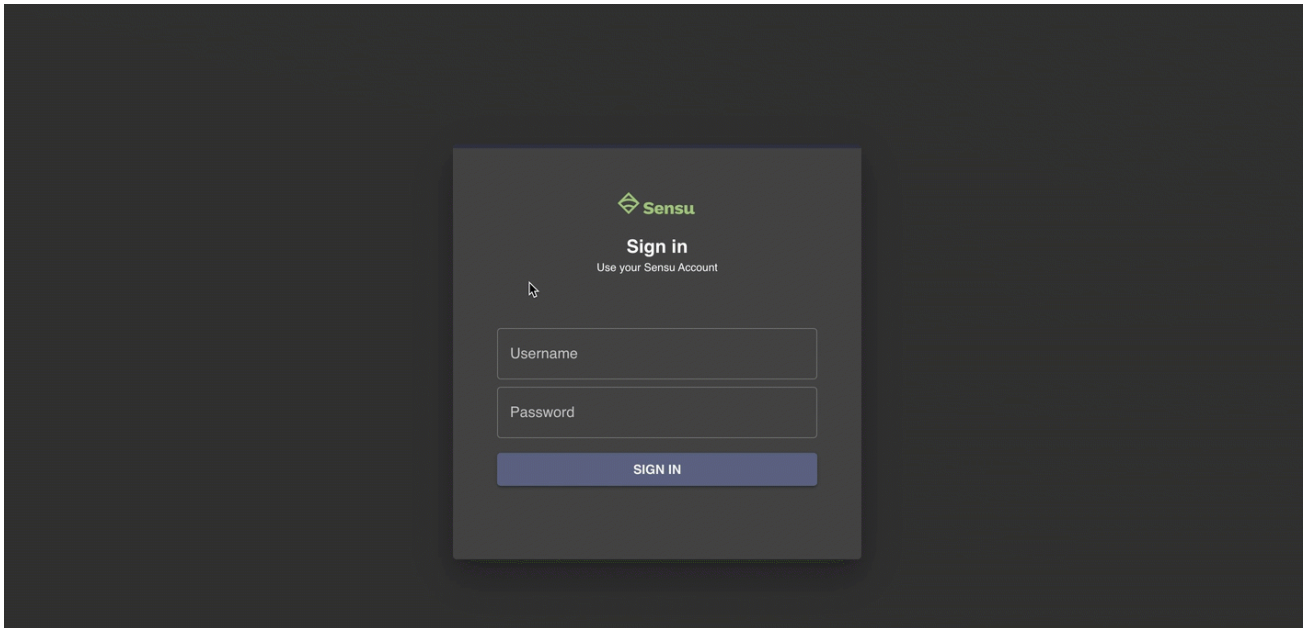
Because the `federation-viewer` user is granted only permissions provided by the built-in `view` role, this user should be able to view all resources across all clusters but should not be able to make any changes. If you haven't changed the permissions of the default `admin` user, that user should be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

Next steps

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

Scale Sensu Go with Enterprise datastore

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

Sensu Go's datastore feature enables scaling your monitoring to many thousands of events per second.

For each unique entity/check pair, Sensu records the latest event object in its datastore. By default, Sensu uses the embedded etcd datastore for event storage. The embedded etcd datastore helps you get started, but as the number of entities and checks in your Sensu implementation grows, so does the rate of events being written to the datastore. In a clustered deployment of etcd, whether embedded or external to Sensu, each event received by a member of the cluster must be replicated to other members, increasing network and disk IO utilization.

Our team documented configuration and testing of Sensu running on bare metal infrastructure in the sensu/sensu-perf project. This configuration comfortably handled 12,000 Sensu agent connections (and their keepalives) and processed more than 8,500 events per second.

This rate of events should be sufficient for many installations but assumes an ideal scenario where Sensu backend nodes use direct-attached, dedicated non-volatile memory express (NVMe) storage and are connected to a dedicated LAN. Deployments on public cloud providers are not likely to achieve similar results due to sharing both disk and network bandwidth with other tenants. Adhering to the cloud provider's recommended practices may also become a factor because many operators are inclined to deploy a cluster across multiple availability zones. In such a deployment cluster, communication happens over shared WAN links, which are subject to uncontrolled variability in throughput and latency.

The Enterprise datastore can help operators achieve much higher rates of event processing and minimize the replication communication between etcd peers. The `sensu-perf` test environment comfortably handles 40,000 Sensu agent connections (and their keepalives) and processes more than 36,000 events per second under ideal conditions.

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)
- ˥ <u>Licensed Sensu Go backend</u>

For optimal performance, we recommend the following PostgreSQL configuration parameters and settings as a starting point for your `postgresql.conf` file:

```
max_connections = 200

shared_buffers = 10GB

maintenance_work_mem = 1GB

vacuum_cost_delay = 10ms
vacuum_cost_limit = 10000

bgwriter_delay = 50ms
bgwriter_lru_maxpages = 1000

max_worker_processes = 8
max_parallel_maintenance_workers = 2
max_parallel_workers_per_gather = 2
max_parallel_workers = 8

synchronous_commit = off

wal_sync_method = fdatasync
wal_writer_delay = 5000ms
max_wal_size = 5GB
min_wal_size = 1GB

checkpoint_completion_target = 0.9

autovacuum_naptime = 10s
autovacuum_vacuum_scale_factor = 0.05
autovacuum_analyze_scale_factor = 0.025
```

Adjust the parameters and settings as needed based on your hardware and the performance you

observe. Read the [PostgreSQL parameters documentation](#) for information about setting parameters.

Configure Postgres

Before Sensu can start writing events to Postgres, you need a database and an account with permissions to write to that database. To provide consistent event throughput, we recommend exclusively dedicating your Postgres instance to storage of Sensu events.

If you have administrative access to Postgres, you can create the database and user.

1. Change to the postgres user and open the Postgres prompt (`postgres=#`):

   ```
   sudo -u postgres psql
   ```

2. Create the `sensu_events` database:

   ```
   CREATE DATABASE sensu_events;
   ```

   Postgres will return a confirmation message: `CREATE DATABASE` .

3. Create the `sensu` role with a password:

   ```
   CREATE USER sensu WITH ENCRYPTED PASSWORD 'mypass';
   ```

   Postgres will return a confirmation message: `CREATE ROLE` .

4. Grant the `sensu` role all privileges for the `sensu_events` database:

   ```
   GRANT ALL PRIVILEGES ON DATABASE sensu_events TO sensu;
   ```

   Postgres will return a confirmation message: `GRANT` .

5. Type `\q` to exit the PostgreSQL prompt.

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:

1. Make a copy of the current `pg_hba.conf` file:

    ```
    sudo cp /var/lib/pgsql/data/pg_hba.conf /var/tmp/pg_hba.conf.bak
    ```

2. Give the Sensu user permissions to connect to the `sensu_events` database from any IP address:

    ```
    echo 'host sensu_events sensu 0.0.0.0/0 md5' | sudo tee -a
    /var/lib/pgsql/data/pg_hba.conf
    ```

3. Restart the postgresql service to activate the `pg_hba.conf` changes:

    ```
    sudo systemctl restart postgresql
    ```

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

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
```

```yaml
  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

```json
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "dsn": "postgresql://sensu:mypass@10.0.2.15:5432/sensu_events",
    "pool_size": 20
  }
}
```

Save this configuration as `my-postgres.yml` or `my-postgres.json` and install it with `sensuctl` :

SHELL

```shell
sensuctl create --file my-postgres.yml
```

SHELL

```shell
sensuctl create --file my-postgres.json
```

The Sensu backend is now configured to use Postgres for event storage!

In the web UI and in sensuctl, event history will appear incomplete. When Postgres configuration is provided and the backend successfully connects to the database, etcd event history is not migrated. New events will be written to Postgres as they are processed, with the Postgres datastore ultimately being brought up to date with the current state of your monitored infrastructure.

Aside from event history, which is not migrated from etcd, there's no observable difference when using Postgres as the event store, and neither interface supports displaying the PostgresConfig type.

To verify that the change was effective and your connection to Postgres was successful, look at the sensu-backend log:

```
{"component":"store","level":"warning","msg":"trying to enable external event store","time":"2019-10-02T23:31:38Z"}
{"component":"store","level":"warning","msg":"switched event store to postgres","time":"2019-10-02T23:31:38Z"}
```

You can also use `psql` to verify that events are being written to the `sensu_events` database.

1. Change to the postgres user and open the Postgres prompt (`postgres=#`):

```
sudo -u postgres psql
```

2. Connect to the `sensu_events` database:

```
\c sensu_events
```

Postgres will return a confirmation message:

```
You are now connected to database "sensu_events" as user "postgres".
```

The prompt will change to `sensu_events=#`.

3. List the tables in the `sensu_events` database:

```
\dt
```

Postgres will list the tables:

```
              List of relations
    Schema |          Name          | Type  | Owner
   --------+------------------------+-------+-------
    public | events                 | table | sensu
    public | migration_version      | table | sensu
   (2 rows)
```

4. Request a list of all entities reporting keepalives:

```
select sensu_entity from events where sensu_check = 'keepalive';
```

Postgres will return a list of the entities:

```
    sensu_entity
   --------------
    i-414141
    i-424242
    i-434343
   (3 rows)
```

Revert to the built-in datastore

If you want to revert to the default etcd event store, delete the PostgresConfig resource. In this example, `my-postgres.yml` or `my-postgres.json` contain the same configuration you used to configure the Enterprise event store earlier in this guide:

SHELL

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

SHELL

```
sensuctl delete --file my-postgres.json
```

To verify that the change was effective, look for messages similar to these in the sensu-backend log:

```
{"component":"store","level":"warning","msg":"store configuration
deleted","store":"/sensu.io/api/enterprise/store/v1/provider/postgres01","time":"201
9-10-02T23:29:06Z"}
{"component":"store","level":"warning","msg":"switched event store to
etcd","time":"2019-10-02T23:29:06Z"}
```

Similar to enabling Postgres, switching back to the etcd datastore does not migrate current observability event data from one store to another.The web UI or sensuctl output may list outdated events until the etcd datastore catches up with the current state of your monitored infrastructure.

Configure Postgres streaming replication

Postgres supports an active standby with streaming replication.Configure streaming replication to replicate all Sensu events written to the primary Postgres server to the standby server.

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

Follow the steps in this section to create and add the replication role, set streaming replication configuration parameters, bootstrap the standby host, and confirm successful Postgres streaming replication.

Create and add the replication role

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

> NOTE: Complete the steps to create and add the replication role on the **primary** Postgres host.

1. Change to the postgres user and open the Postgres prompt (`postgres=#`):

   ```
   sudo -u postgres psql
   ```

2. Create the `repl` role:

```
CREATE ROLE repl PASSWORD '<your-password>' LOGIN REPLICATION;
```

Postgres will return a confirmation message: `CREATE ROLE` .

3. Type `\q` to exit the PostgreSQL prompt.

4. Add the replication role to `pg_hba.conf` using an md5-encrypted password. Make a copy of the current `pg_hba.conf` :

```
sudo cp /var/lib/pgsql/data/pg_hba.conf /var/tmp/pg_hba.conf.bak
```

5. In the following command, replace `<standby_ip>` with the IP address of your standby Postgres host and run the command:

```
export STANDBY_IP=<standby-ip>
```

6. Give the repl user permissions to replicate from the standby host:

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

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

```
sudo systemctl restart postgresql
```

Set streaming replication configuration parameters

NOTE: Complete the steps to set streaming replication configuration parameters on the **primary**

1. Make a copy of the `postgresql.conf` :

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

2. Append the necessary configuration options.

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

3. Set the maximum number of concurrent connections from the standby servers:

   ```
   echo 'max_wal_senders = 5' | sudo tee -a /var/lib/pgsql/data/postgresql.conf
   ```

4. To prevent the primary server from removing the WAL segments required for the standby server before shipping them, set the minimum number of segments retained in the `pg_xlog` directory:

   ```
   echo 'wal_keep_segments = 32' | sudo tee -a
   /var/lib/pgsql/data/postgresql.conf
   ```

   At minimum, the number of `wal_keep_segments` should be larger than the number of segments generated between the beginning of online backup and the startup of streaming replication.

   > **NOTE**: *If you enable WAL archiving to an archive directory accessible from the standby, this step may not be necessary.*

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

   ```
   sudo systemctl restart postgresql
   ```

Bootstrap the standby host

1. If the standby host has ever run Postgres, stop Postgres and empty the data directory:

   ```
   sudo systemctl stop postgresql
   ```

   ```
   sudo mv /var/lib/pgsql/data /var/lib/pgsql/data.bak
   ```

2. Make the standby data directory:

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

3. In the following command, replace `<primary_ip>` with the IP address of your primary Postgres host and run the command:

   ```
   export PRIMARY_IP=<primary_ip>
   ```

4. Bootstrap the standby data directory:

   ```
   sudo -u postgres pg_basebackup -h $PRIMARY_IP -D /var/lib/pgsql/data -P -U
   repl -R --wal-method=stream
   ```

5. Enter your password at the Postgres prompt:

   ```
   Password:
   ```

After you enter your password, Postgres will list database copy progress:

```
30318/30318 kB (100%), 1/1 tablespace
```

Confirm replication

1. On the **primary** Postgres host, remove primary-only configurations:

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

2. Start the PostgreSQL service:

    ```
    sudo systemctl start postgresql
    ```

3. Check the primary host commit log location:

    ```
    sudo -u postgres psql -c "select pg_current_wal_lsn()"
    ```

    Postgres will list the primary host commit log location:

    ```
     pg_current_wal_lsn
    --------------------------
     0/3000568
    (1 row)
    ```

4. On the **standby** Postgres host, check the commit log location:

    ```
    sudo -u postgres psql -c "select pg_last_wal_receive_lsn()"
    ```

```
sudo -u postgres psql -c "select pg_last_wal_replay_lsn()"
```

Postgres will list the standby host commit log location:

```
 pg_last_wal_receive_lsn
-----------------------------
 0/3000568
(1 row)
```

```
 pg_last_wal_replay_lsn
-----------------------------
 0/3000568
(1 row)
```

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

Datastore reference

Sensu stores the most recent event for each entity and check pair using either an etcd (default) or PostgreSQL database. You can access observability event data with the Sensu web UI Events page, `sensuctl event` commands, and the events API. For longer retention of observability event data, integrate Sensu with a time-series database like InfluxDB or a searchable index like ElasticSearch or Splunk.

Use default event storage

By default, Sensu uses its embedded etcd database to store configuration and event data. This embedded database allows you to get started with Sensu without deploying a complete, scalable architecture. Sensu's default embedded etcd configuration listens for unencrypted communication on ports 2379 (client requests) and 2380 (peer communication).

Sensu can be configured to disable the embedded etcd database and use one or more external etcd nodes for configuration and event storage instead. To stand up an external etcd cluster, follow etcd's clustering guide using the same store configuration. Do not configure external etcd in Sensu via backend command line flags or the backend configuration file (`/etc/sensu/backend.yml`).

As your deployment grows beyond the proof-of-concept stage, review Deployment architecture for Sensu for more information about deployment considerations and recommendations for a production-ready Sensu deployment.

Sensu requires at least etcd 3.3.2 and is tested against releases in the 3.3.x series. etcd version 3.4.0 is compatible with Sensu but may result in slower performance than the 3.3.x series.

Scale event storage

COMMERCIAL FEATURE: Access enterprise-scale event storage in the packaged Sensu Go distribution. For more information, read Get started with commercial features.

Sensu supports using an external PostgreSQL instance for event storage in place of etcd. PostgreSQL can handle significantly higher volumes of Sensu events, which allows you to scale Sensu beyond

etcd's 8GB limit.

When configured with a PostgreSQL event store, Sensu connects to PostgreSQL to store and retrieve event data in place of etcd.Etcd continues to store Sensu entity and configuration data.You can access event data stored in PostgreSQL using the same Sensu web UI, API, and sensuctl processes as etcd-stored events.

PostgreSQL requirements

Sensu supports PostgreSQL 9.5 and later, including Amazon Relational Database Service (Amazon RDS) when configured with the PostgreSQL engine.Read the PostgreSQL documentation to install and configure PostgreSQL.

For optimal performance, we recommend the following PostgreSQL configuration parameters and settings as a starting point for your `postgresql.conf` file:

```
max_connections = 200

shared_buffers = 10GB

maintenance_work_mem = 1GB

vacuum_cost_delay = 10ms
vacuum_cost_limit = 10000

bgwriter_delay = 50ms
bgwriter_lru_maxpages = 1000

max_worker_processes = 8
max_parallel_maintenance_workers = 2
max_parallel_workers_per_gather = 2
max_parallel_workers = 8

synchronous_commit = off

wal_sync_method = fdatasync
wal_writer_delay = 5000ms
max_wal_size = 5GB
min_wal_size = 1GB
```

```
checkpoint_completion_target = 0.9

autovacuum_naptime = 10s
autovacuum_vacuum_scale_factor = 0.05
autovacuum_analyze_scale_factor = 0.025
```

Adjust the parameters and settings as needed based on your hardware and the performance you observe.Read the PostgreSQL parameters documentation for information about setting parameters.

Configure the PostgreSQL event store

At the time when you enable the PostgreSQL event store, event data cuts over from etcd to PostgreSQL.This results in a loss of recent event history.No restarts or Sensu backend configuration changes are required to enable the PostgreSQL event store.

When you successfully enable PostgreSQL as the Sensu Go event store, the Sensu backend log will include a message like this:

```
Mar 10 17:44:45 sensu-centos sensu-backend[1365]: {"component":"store-
providers","level":"warning","msg":"switched event store to postgres","time":"2020-
03-10T17:44:45Z"}
```

After you install and configure PostgreSQL, configure Sensu by creating a `PostgresConfig` resource like the following example.Review the datastore specification for more information.

YML

```
---
type: PostgresConfig
api_version: store/v1
metadata:
  name: my-postgres
spec:
  batch_buffer: 0
  batch_size: 1
  batch_workers: 0
  dsn: "postgresql://user:secret@host:port/dbname"
  max_conn_lifetime: 5m
```

```
      max_idle_conns: 2
      pool_size: 20
      strict: true
```

JSON

```json
{
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true
  }
}
```

Save your `PostgresConfig` resource definition to a file (in this example, `postgres.yml` or `postgres.json`).Then, use sensuctl configured as the admin user to activate the PostgreSQL event store.

SHELL

```shell
sensuctl create --file postgres.yml
```

SHELL

```shell
sensuctl create --file postgres.json
```

To update your Sensu PostgreSQL configuration, repeat the `sensuctl create` process.You can expect PostgreSQL status updates in the Sensu backend logs at the `warn` log level and PostgreSQL

error messages in the Sensu backend logs at the `error` log level.

Disable the PostgreSQL event store

To disable the PostgreSQL event store, use `sensuctl delete` with your `PostgresConfig` resource definition file:

SHELL

```
sensuctl delete --file postgres.yml
```

SHELL

```
sensuctl delete --file postgres.json
```

The Sensu backend log will include a message to record that you successfully disabled PostgreSQL as the Sensu Go event store:

```
Mar 10 17:35:04 sensu-centos sensu-backend[1365]: {"component":"store-providers","level":"warning","msg":"switched event store to etcd","time":"2020-03-10T17:35:04Z"}
```

When you disable the PostgreSQL event store, event data cuts over from PostgreSQL to etcd, which results in a loss of recent event history.No restarts or Sensu backend configuration changes are required to disable the PostgreSQL event store.

Datastore specification

Top-level attributes

type

| description | Top-level attribute that specifies the `sensuctl create` resource type. PostgreSQL datastore configs should always be type `PostgresConfig`. |

required	true
type	String

example

YML

```yaml
type: PostgresConfig
```

JSON

```json
{
  "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

YML

```yaml
api_version: store/v1
```

JSON

```json
{
  "api_version": "store/v1"
}
```

metadata

description	Top-level scope that contains the PostgreSQL datastore `name` and `created_by` field.
required	true
type	Map of key-value pairs **YML**
example	

```yaml
metadata:
  name: my-postgres
  created_by: admin
```

JSON

```json
{
  "metadata": {
    "name": "my-postgres",
    "created_by": "admin"
  }
}
```

spec

description	Top-level map that includes the PostgreSQL datastore config spec attributes.
required	true
type	Map of key-value pairs **YML**
example	

```yaml
spec:
  batch_buffer: 0
  batch_size: 1
  batch_workers: 0
  dsn: 'postgresql://user:secret@host:port/dbname'
  max_conn_lifetime: 5m
  max_idle_conns: 2
```

```
      pool_size: 20
      strict: true
```

JSON

```json
{
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true
  }
}
```

Metadata attributes

name
description	PostgreSQL datastore name used internally by Sensu.
required	true
type	String

example	**YML**
	```name: my-postgres```
	
	JSON
	``json
	{
	  "name": "my-postgres"
	}
	``

created_by

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

Spec attributes

batch_buffer

description	Maximum number of requests to buffer in memory. **WARNING**: The batcher is sensitive to configuration values, and some `batch_buffer`, `batch_size`, and `batch_workers` combinations will not work optimally. We do not recommend configuring this attribute while we are testing and improving it.
required	false

default	0
type	Integer

example	**YML**

```yml
batch_buffer: 0
```

JSON

```json
{
  "batch_buffer": 0
}
```

batch_size

description	Number of requests in each PostgreSQL write transaction, as specified in the PostgreSQL configuration.

> **WARNING**: *The batcher is sensitive to configuration values, and some `batch_buffer`, `batch_size`, and `batch_workers` combinations will not work optimally. We do not recommend configuring this attribute while we are testing and improving it.*

required	false
default	1
type	Integer

example	**YML**

```yml
batch_size: 1
```

JSON

```json
{
  "batch_size": 1
```

```
  }
```

batch_workers

description	Number of Goroutines sending data to PostgreSQL, as specified in the PostgreSQL configuration.

> **WARNING**: The batcher is sensitive to configuration values, and some `batch_buffer`, `batch_size`, and `batch_workers` combinations will not work optimally. We do not recommend configuring this attribute while we are testing and improving it.

required	false
default	Current PostgreSQL pool size
type	Integer

YML

example

```yml
batch_workers: 0
```

JSON

```json
{
  "batch_workers": 0
}
```

dsn

description	Data source names. Specified as a URL or PostgreSQL connection string. The Sensu backend uses the Go pq library, which supports a subset of the PostgreSQL libpq connection string parameters.

required	true
type	String

example	YML

```yml
dsn: 'postgresql://user:secret@host:port/dbname'
```

JSON

```json
{
  "dsn": "postgresql://user:secret@host:port/dbname"
}
```

max_conn_lifetime

description	Maximum time a connection can persist before being destroyed. Specify values with a numeral and a letter indicator: `s` to indicate seconds, `m` to indicate minutes, and `h` to indicate hours. For example, `1m`, `2h`, and `2h1m3s` are valid.
required	false
type	String

example	YML

```yml
max_conn_lifetime: 5m
```

JSON

```json
{
  "max_conn_lifetime": "5m"
}
```

max_idle_conns

description	Maximum number of number of idle connections to retain.
required	false
default	`2`
type	Integer

example

YML

```yml
max_idle_conns: 2
```

JSON

```json
{
  "max_idle_conns": 2
}
```

pool_size

description	Maximum number of connections to hold in the PostgreSQL connection pool. We recommend `20` for most instances.
required	false
default	`0` (unlimited)
type	Integer

example

YML

```yml
pool_size: 20
```

JSON

```json
{
  "pool_size": 20
}
```

strict	
description	If `true` , when the PostgresConfig resource is created, configuration validation will include connecting to the PostgreSQL database and executing a query to confirm whetherthe connected user has permission to create database tables. Otherwise, `false` . We recommend setting `strict: true` in most cases. If the connection fails or the user does not have permission to create database tables, resource configuration will fail and the configuration will not be persisted. This extended configuration is useful for debugging when you are not sure whether the configuration is correct or the database is working properly.
required	false
default	false
type	Boolean

example	**YML**
	 ``
	   strict: true
	``
	
	 JSON
	 ``
	  {
	     "strict": true
	  }
	``

Etcd replicators reference

NOTE: *EtcdReplicator 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 EtcdReplicator datatype will not use a namespace because it applies cluster-wide. Therefore, only cluster role RBAC bindings will apply to it.

Etcd replicator examples

Use the following four examples for `Role`, `RoleBinding`, `ClusterRole`, and `ClusterRoleBinding` resources to create a full replication of [RBAC policy](#).

NOTE: *If you do not specify a namespace when you create a replicator, all namespaces for a given resource are replicated.*

`Role` resource example

YML

```
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: role_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
```

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

    insecure: false

    url: http://127.0.0.1:2379

    api_version: core/v2

    resource: Role

    replication_interval_seconds: 30
```

JSON

```json
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "role_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

`RoleBinding` resource example

YML

```yaml
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: rolebinding_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  key: /path/to/ssl/key.pem
  insecure: false
```

```
    url: http://127.0.0.1:2379
    api_version: core/v2
    resource: RoleBinding
    replication_interval_seconds: 30
```

JSON

```json
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "rolebinding_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "RoleBinding",
    "replication_interval_seconds": 30
  }
}
```

`ClusterRole` resource example

YML

```yaml
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: clusterrole_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  key: /path/to/ssl/key.pem
  insecure: false
  url: http://127.0.0.1:2379
  api_version: core/v2
```

```
    resource: ClusterRole
    replication_interval_seconds: 30
```

JSON

```json
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "clusterrole_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "ClusterRole",
    "replication_interval_seconds": 30
  }
}
```

`ClusterRoleBinding` resource example

YML

```yaml
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: clusterrolebinding_replicator
spec:
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  key: /path/to/ssl/key.pem
  insecure: false
  url: http://127.0.0.1:2379
  api_version: core/v2
  resource: Role
  replication_interval_seconds: 30
```

JSON

```json
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "clusterrolebinding_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "ClusterRoleBinding",
    "replication_interval_seconds": 30
  }
}
```

Critical success factors for etcd replication

Before you implement etcd replicators, review these details — they are critical to your success.

Bind your etcd listener to an external port that is *not* the default.

- ¬ Replication will not work if you bind your etcd listener to the default port.

Use only addresses that clients can route to for `etcd-client-advertise-urls`.

- ¬ If you use addresses that clients cannot route to for `etcd-client-advertise-urls`, replication may be inconsistent: it may work at first but then stop working later.

Put the certificate and key of the follower cluster in files that the leader can access.

- ¬ If the leader cannot access the follower cluster files that contain the certificate and key, replication will not work.

For self-signed certificates, supply the CA certificate in the replicator definition.

- ¬ If you have a self-signed certificate and you do not supply the CA certificate in the replicator definition, replication will not work.

If you're using insecure mode, use TLS mutual authentication.

- ¬ Never use insecure mode without TLS mutual authentication outside of a testbed.

Create a replicator for each resource type you want to replicate.

- ¬ Replicating `namespace` resources will **not** replicate the resources that belong to those namespaces.

> **WARNING**: Make sure to confirm your configuration. The server will accept incorrect EtcdReplicator definitions without sending a warning. If your configuration is incorrect, replication will not work.

Create a replicator

You can use the the federation API directly or `sensuctl create` to create replicators.

When you create or update a replicator, an entry is added to the store and a new replicator process will spin up. The replicator process watches the keyspace of the resource to be replicated and replicates all keys to the specified cluster in a last-write-wins fashion.

When the cluster starts up, each sensu-backend scans the stored replicator definitions and starts a replicator process for each replicator definition. Source clusters with more than one sensu-backend will cause redundant writes. This is harmless, but you should consider it when designing a replicated system.

> **NOTE**: Create a replicator for each resource type you want to replicate. Replicating `namespace` resources will **not** replicate the resources that belong to those namespaces.

Delete a replicator

When you delete a replicator, the replicator will issue delete events to the remote cluster for all of the

keys in its prefix.It will not issue a delete of the entire key prefix (just in case the prefix is shared by keys that are local to the remote cluster).

Rather than altering an existing replicator's connection details, delete and recreate the replicator with the new connection details.

Replicator configuration

Etcd replicators are etcd key space replicators.Replicators contain configuration for forwarding a set of keys from one etcd cluster to another.Replicators are configured by specifying the TLS details of the remote cluster, its URL, and a resource type.

Etcd replicator specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Always `EtcdReplicator.`
required	true
type	String

example	YML
	``yml
	type: EtcdReplicator
	``
	JSON
	``json
	{
	  "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

YML

example

```yaml
api_version: federation/v1
```

JSON

```json
{
  "api_version": "federation/v1"
}
```

metadata

description	Top-level scope that contains the replicator `name` and `created_by` value. Namespace is not supported in the metadata because etcd replicators are cluster-wide resources.
required	true
type	Map of key-value pairs

YML

example

```yaml
metadata:
  name: my_replicator
  created_by: admin
```

JSON

```json
{
  "metadata": {
    "name": "my_replicator",
```

```
      "created_by": "admin"
    }
  }
```

spec

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

```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: Role
  replication_interval_seconds: 30
```

JSON

```json
{
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://127.0.0.1:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}
```

Metadata attributes

name	
description	Replicator name used internally by Sensu.
required	true
type	String
example	**YML**

```yml
name: my_replicator
```

JSON

```json
{
  "name": "my_replicator"
}
```

created_by	
description	Username of the Sensu user who created the replicator or last updated the replicator. Sensu automatically populates the `created_by` field when the replicator is created or updated.
required	false
type	String
example	**YML**

```yml
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

Spec attributes

ca_cert	
description	Path to the PEM-format CA certificate to use for TLS client authentication.
required	true if `insecure: false` (the default configuration). If `insecure: true`, `ca_cert` is not required.
type	String

YML

```
ca_cert: /path/to/trusted-certificate-authorities.pem
```

JSON

```
{
  "ca_cert": "/path/to/trusted-certificate-authorities.pem"
}
```

cert	
description	Path to the PEM-format certificate to use for TLS client authentication. This certificate is required for secure client communication.
required	true if `insecure: false` (the default configuration). If `insecure: true`, `cert` is not required.
type	String

example	**YML**
| | ```yaml
cert: /path/to/ssl/cert.pem
``` |
| | **JSON** |
| | ```json
{
  "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. This key and its corresponding certificate are required for secure client communication.
required	true if `insecure: false` (the default configuration). If `insecure: true`, `key` is not required.
type	String
example	**YML**
| | ```yaml
key: /path/to/ssl/key.pem
``` |
| | **JSON** |
| | ```json
{
  "key": "/path/to/ssl/key.pem"
}
``` |

insecure

description	`true` to disable transport security. Otherwise, `false`.

> ***WARNING***: *Disable transport security with care.*

required	false
type	Boolean
default	`false`

example

YML

```
insecure: false
```

JSON

```
{
  "insecure": false
}
```

url

description	Destination cluster URL. If specifying more than one, use a comma to separate. Replace with a non-default value for secure client communication.
required	true
type	String

example

YML

```
url: http://127.0.0.1:2379
```

JSON

```
{
  "url": "http://127.0.0.1:2379"
}
```

api_version

description	Sensu API version of the resource to replicate.
required	false
type	String
default	`core/v2`
| example | **YML**<br><br>```yaml
api_version: core/v2
```<br><br>**JSON**<br><br>```json
{
  "api_version": "core/v2"
}
``` |

resource

description	Name of the resource to replicate.
required	true
type	String
| example | **YML**<br><br>```yaml
resource: Role
```<br><br>**JSON**<br><br>```json
{
  "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

YML

example

```yaml
namespace: default
```

JSON

```json
{
  "namespace": "default"
}
```

replication_interval_seconds

description	Interval at which the resource will be replicated. In seconds.
required	false
type	Integer
default	30

YML

example

```yaml
replication_interval_seconds: 30
```

JSON

```json
{
```

```
    "replication_interval_seconds": 30
}
```

Control Access

Sensu administrators control access by authentication and authorization.

Authentication verifies user identities to confirm that users are who they say they are.Sensu requires username and password authentication to access the web UI, API, and sensuctl command line tool.You can use Sensu's built-in basic authentication provider or configure external authentication providers.

> **NOTE**: For API-specific authentication, read the API overview and Use API keys to authenticate to Sensu.

Authorization establishes and manages user permissions: the extent of access users have for different Sensu resources.Configure authorization with role-based access control (RBAC) to exercise fine-grained control over how they interact with Sensu resources.

Authentication

Sensu web UI and sensuctl command line tool users can authenticate via built-in basic authentication provider or Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) when the administrator configures external single sign-on (SSO) authentication providers.

Sensu agents authenticate to the Sensu backend using either basic or mutual transport layer security (TLS) authentication.

Use built-in basic authentication

Sensu's built-in basic authentication 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.

Sensu hashes user passwords using the bcrypt algorithm and records the basic authentication credentials in etcd.

Use a single sign-on (SSO) authentication provider

In addition to built-in basic authentication, Sensu includes commercial support for single sign-on (SSO) authentication using external authentication providers via Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).

Read Configure single sign-on (SSO) authentication for general information about configuring an SSO authentication provider. Read the LDAP, AD, or OIDC reference documentation for provider-specific information.

Authorization

After you set up authentication, configure authorization via role-based access control (RBAC) to give those users permissions within Sensu. RBAC allows you to specify actions users are allowed to take against resources, within namespaces or across all namespaces, based on roles bound to the user or to one or more groups the user is a member of. Read Create a read-only user for an example.

- **Namespaces** partition resources within Sensu. Sensu entities, checks, handlers, and other namespaced resources belong to a single namespace.

- **Roles** create sets of permissions (like GET and DELETE) tied to resource types. **Cluster roles** apply permissions across all namespaces and may include access to cluster-wide resources like users and namespaces.

- **Role bindings** assign a role to a set of users and groups within a namespace. **Cluster role bindings** assign a cluster role to a set of users and groups across all namespaces.

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

After you configure an authentication provider and establish the roles and bindings to grant authenticated users the desired privileges, those users can log in via sensuctl and the web UI using a single-sign-on username and password. Users do _not_ need to provide the username prefix for the authentication provider when logging in to Sensu.

Configure single sign-on (SSO) authentication

> ***COMMERCIAL FEATURE***: *Access authentication providers for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read [Get started with commercial features](#).*

Sensu requires username and password authentication to access the [web UI](#), [API](#), and [sensuctl](#) command line tool.

In addition to the [built-in basic authentication provider](#), Sensu offers [commercial support](#) for using Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC) for single sign-on (SSO) authentication.

This guide describes general information for configuring an authentication provider for SSO. Read the [LDAP](#), [AD](#), or [OIDC](#) reference documentation for provider-specific examples and specifications.

Configure authentication providers

To configure an external authentication provider for SSO, first write an authentication provider configuration definition. Follow the examples and specifications for your provider:

- **Lightweight Directory Access Protocol (LDAP)**, including standards-compliant tools like OpenLDAP ([configuration examples](#) and [specification](#))

- **Microsoft Active Directory (AD)**, including Azure AD ([configuration examples](#) and [specification](#))

- **OpenID Connect 1.0 protocol (OIDC)**, including tools like Okta and PingFederate ([configuration examples](#) and [specification](#))

Save your configuration definition to a file, such as `authconfig.yaml` or `authconfig.json`.

After you have a saved configuration definition, you can apply the configuration with sensuctl. Log in to sensuctl as the [default admin user](#) and use sensuctl to apply your authentication provider configuration to Sensu:

```shell
sensuctl create --file authconfig.yml
```

```shell
sensuctl create --file authconfig.json
```

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

```shell
sensuctl auth list
```

The response will list your authentication provider types and names:

```
Type    Name
──────  ──────────
ldap   openldap
```

Manage authentication providers

View and delete authentication providers with the authentication providers API or these sensuctl commands.

To view active authentication providers:

```shell
sensuctl auth list
```

To view configuration details for an authentication provider named `openldap`:

```shell
sensuctl auth info openldap
```

To delete an authentication provider named `openldap`:

```
sensuctl auth delete openldap
```

Use API keys to authenticate to Sensu

The Sensu API key feature (core/v2.APIKey) is a persistent universally unique identifier (UUID) that maps to a stored Sensu username. The advantages of authenticating with API keys rather than <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 use sensuctl to generate a new API key for the admin user, run:

```
sensuctl api-key grant admin
```

The response will include the new API key:

```
Created: /api/core/v2/apikeys/7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2
```

To get information about an API key:

SHELL

```
sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format yaml
```

SHELL

```
sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format wrapped-json
```

SHELL

```
sensuctl api-key info 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --format json
```

The response will include information about the API key in the specified format:

YML

```yaml
---
type: APIKey
api_version: core/v2
metadata:
  created_by: admin
  name: 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2
spec:
  created_at: 1570718917
  username: admin
```

SHELL

```json
{
  "type": "APIKey",
  "api_version": "core/v2",
  "metadata": {
```

```
      "name": "7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2",
      "created_by": "admin"
    },
    "spec": {
      "created_at": 1570718917,
      "username": "admin"
    }
  }
```

JSON

```
{
  "metadata": {
    "name": "7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2",
    "created_by": "admin"
  },
  "username": "admin",
  "created_at": 1570718917
}
```

To get a list of all API keys:

```
sensuctl api-key list
```

The response lists all API keys along with the name of the user who created each key and the date
and time each key was created:

```
       Name                 Username      Created At
 _____ _____ _____
  7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2   admin     2019-10-10 14:48:37 -0700 PDT
```

To revoke an API key for the admin user:

```
sensuctl api-key revoke 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2 --skip-confirm
```

The response will confirm that the API key is deleted:

```
Deleted
```

Create a read-only user with role-based access control (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
   ```

   This command creates the following user:

   YML

   ```
   username: alice
   groups:
   - ops
   disabled: false
   ```

   JSON

```
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

2. Create a `read-only` role with `get` and `list` permissions for all resources (`*`) within the `default` namespace:

```
sensuctl role create read-only --verb=get,list --resource=* --namespace=default
```

This command creates the following role resource definition:

YML

```
---
type: Role
api_version: core/v2
metadata:
  created_by: admin
  name: read-only
  namespace: default
spec:
  rules:
  - resource_names: null
    resources:
    - '*'
    verbs:
    - get
    - list
```

JSON

```
{
```

```json
    "type": "Role",
    "api_version": "core/v2",
    "metadata": {
      "created_by": "admin",
      "name": "read-only",
      "namespace": "default"
    },
    "spec": {
      "rules": [
        {
          "resource_names": null,
          "resources": [
            "*"
          ],
          "verbs": [
            "get",
            "list"
          ]
        }
      ]
    }
  }
```

3. Create an `ops-read-only` role binding to assign the `read-only` role to the `ops` group:

```
sensuctl role-binding create ops-read-only --role=read-only --group=ops
```

This command creates the following role binding resource definition:

YML

```yaml
---
type: RoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: ops-read-only
  namespace: default
spec:
```

```
  role_ref:
    name: read-only
    type: Role
  subjects:
  - name: ops
    type: Group
```

JSON

```json
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "ops-read-only",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "read-only",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

All users in the `ops` group now have read-only access to all resources within the default namespace. You can also use role bindings to tie roles directly to users using the `--user` flag.

To manage your RBAC configuration, use the `sensuctl user`, `sensuctl role`, and `sensuctl role-binding` commands.

Create a cluster-wide event-reader user

Suppose you want to create a user with read-only access to events across all namespaces.Because you want this role to have cluster-wide permissions, you'll need to create a **cluster role** and a **cluster role binding**.

1. Create a user with the username `bob` and assign them to the group `ops` :

```
sensuctl user create bob --password='password' --groups=ops
```

This command creates the following user:

YML

```yml
username: bob
groups:
- ops
disabled: false
```

JSON

```json
{
   "username": "bob",
   "groups": [
      "ops"
   ],
   "disabled": false
}
```

2. Create a `global-event-reader` cluster role with `get` and `list` permissions for `events` across all namespaces:

```
sensuctl cluster-role create global-event-reader --verb=get,list --resource=events
```

This command creates the following cluster role resource definition:

YML

```yaml
---
type: ClusterRole
api_version: core/v2
metadata:
  created_by: admin
  name: global-event-reader
spec:
  rules:
  - resource_names: null
    resources:
    - events
    verbs:
    - get
    - list
```

JSON

```json
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "global-event-reader"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "events"
        ],
        "verbs": [
          "get",
          "list"
        ]
      }
    ]
  }
}
```

3. Create an `ops-event-reader` cluster role binding to assign the `global-event-reader` role to the `ops` group:

```
sensuctl cluster-role-binding create ops-event-reader --cluster-role=global-
event-reader --group=ops
```

This command creates the following cluster role binding resource definition:

YML

```yaml
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: ops-event-reader
spec:
  role_ref:
    name: global-event-reader
    type: ClusterRole
  subjects:
  - name: ops
    type: Group
```

JSON

```json
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "ops-event-reader"
  },
  "spec": {
    "role_ref": {
      "name": "global-event-reader",
      "type": "ClusterRole"
    },
    "subjects": [
      {
```

```
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

All users in the `ops` group now have read-only access to events across all namespaces.

Next steps

Now that you know how to create a user, a role, and a role binding to assign a role to a user, check out the RBAC reference for in-depth documentation on role-based access control, examples, and information about cluster-wide permissions.

Read about monitoring as code with Sensu and learn how to use SensuFlow to synchronize your monitoring and observability code with your Sensu deployments.

Create limited service accounts with role-based access control (RBAC)

In some cases, you may want to allow an application or service to interact with Sensu resources. Use Sensu's role-based access control (RBAC) to create and configure accounts that represent applications or services rather than individual human users. These limited service accounts give you fine-grained control of the access and permissions the application or service needs.

For example, you might develop a service that displays a high-level view of your webserver statuses based on an aggregate check. The service itself only needs an API key and permission to read the results of checks executed on your webservers so it can route the check results to the status display. No human user needs to log into the service, and the service does not need edit or delete permissions. A limited service account can provide only the necessary access and permissions.

Limited service accounts are also useful for performing automated processes. This guide explains how to create a limited service account to use with the Sensu EC2 Handler integration to automatically remove AWS EC2 instances that are not in a pending or running state.

By default, Sensu includes a `default` namespace and an `admin` user with full permissions to create, modify, and delete resources within Sensu, including the RBAC resources required to configure a limited service account. This guide requires a running Sensu backend and a sensuctl instance configured to connect to the backend as the `admin` user.

Create a limited service account

A limited service account requires:

- ¬ A user.

- ¬ A role with get, list, and delete permissions for resources within the `default` namespace.

- ¬ A role binding that ties the role to the user.

- ¬ An API key for the user.

> **NOTE**: To use a service account to manage Sensu resources in more than one namespace, create a cluster role instead of a role and a cluster role binding instead of a role binding.

1. Create a user with the username `ec2-service` and a dynamically created random password:

```
sensuctl user create ec2-service --password=$(head -c1M /dev/urandom |
sha512sum | cut -d' ' -f1 | head -c 32)
```

This command creates the following user definition:

YML

```yml
---
type: User
api_version: core/v2
metadata:
  name: ec2-service
spec:
  disabled: false
  username: ec2-service
```

JSON

```json
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {
    "name": "ec2-service"
  },
  "spec": {
    "disabled": false,
    "username": "ec2-service"
  }
}
```

2. Create a `ec2-delete` role with get, list, and delete permissions for entity resources within the `default` namespace:

```
sensuctl role create ec2-delete --verb get,list,delete --resource entities --
```

```
namespace default
```

This command creates the role that has the permissions your service account will need:

YML

```yaml
---
type: Role
api_version: core/v2
metadata:
  created_by: admin
  name: ec2-delete
  namespace: default
spec:
  rules:
  - resource_names: null
    resources:
    - entities
    verbs:
    - get
    - list
    - delete
```

JSON

```json
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "ec2-delete",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "entities"
        ],
        "verbs": [
```

```
          "get",
          "list",
          "delete"
        ]
      }
    ]
  }
}
```

3. Create an `ec2-service-delete` role binding to assign the `ec2-delete` role to the `ec2-service` user:

```
sensuctl role-binding create ec2-service-delete --role ec2-delete --user ec2-service
```

This command creates the role binding that ties the correct permissions (via the `ec2-delete` role) with your service account (via the user `ec2-service`):

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: ec2-service-delete
  namespace: default
spec:
  role_ref:
    name: ec2-delete
    type: Role
  subjects:
  - name: ec2-service
    type: User
```

JSON

```
{
```

```
      "type": "RoleBinding",
      "api_version": "core/v2",
      "metadata": {
        "created_by": "admin",
        "name": "ec2-service-delete",
        "namespace": "default"
      },
      "spec": {
        "role_ref": {
          "name": "ec2-delete",
          "type": "Role"
        },
        "subjects": [
          {
            "name": "ec2-service",
            "type": "User"
          }
        ]
      }
    }
```

4. Create an API key for the `ec2-service` user:

   ```
   sensuctl api-key grant ec2-service
   ```

   The response will include an API key that is assigned to the `ec2-service` user, which you will need to configure the EC2 handler.

The `ec2-service` limited service account is now ready to use with the Sensu EC2 Handler integration.

Add the EC2 handler dynamic runtime asset

To power the handler to remove AWS EC2 instances, use sensuctl to add the Sensu Go EC2 Handler dynamic runtime asset:

```
sensuctl asset add sensu/sensu-ec2-handler:0.4.0
```

The response will indicate that the asset was added:

```
fetching bonsai asset: sensu/sensu-ec2-handler:0.4.0
added asset: sensu/sensu-ec2-handler:0.4.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu/sensu-ec2-handler"].
```

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

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read *the asset reference* for more information about dynamic runtime asset builds.

Configure an EC2 handler for the service account

To configure the EC2 handler, you will need AWS account credentials and details for the AWS instance you want to manage, like the AWS instance ID.You will also need the API key for the `ec2-service` user.

> **NOTE**: Use *secrets management* to configure environment variables for your AWS access and secret keys and the `ec2-service` user's API key.Do not expose this sensitive information by listing it directly in the handler definition.
>
> The *Sensu Go EC2 Handler's Bonsai page* includes an example for configuring secrets definitions with Sensu's built-in `env` *secrets provider*.

In the following code, replace these bracketed placeholders with valid values:

- `<AWS_REGION>` : the AWS region where your EC2 instance is located.

- `<AWS_INSTANCE_ID_LABEL>` : the Sensu entity label that contains the AWS instance ID. If your AWS EC2 instance entities do not include labels that specify the instance ID, use the `aws-instance-id` attribute instead and enter the AWS instance ID itself as the value.

- `<http://localhost:8080>` : the Sensu API URL.

You can also adjust the `aws-allowed-instance-states` value to include any of the Sensu EC2 integration's underlined available states. This example lists only "pending" and "running."

Then, run this command with your valid values in place to create the handler defintion:

TEXT

```
cat << EOF | sensuctl create
---
type: Handler
api_version: core/v2
metadata:
  namespace: default
  name: sensu-ec2-handler
spec:
  type: pipe
  runtime_assets:
    - sensu/sensu-ec2-handler
  filters:
    - is_incident
    - not_silenced
  command: >-
    sensu-ec2-handler
    --aws-region <AWS_REGION>
    --aws-instance-id-label <AWS_INSTANCE_ID_LABEL>
    --aws-allowed-instance-states pending,running
    --sensu-api-url <http://localhost:8080>
  secrets:
    - name: AWS_ACCESS_KEY_ID
      secret: <YOUR_AWS_ACCESS_KEY_ID>
    - name: AWS_SECRET_KEY
      secret: <YOUR_AWS_SECRET_KEY>
    - name: SENSU_API_KEY
      secret: <YOUR_SENSU_API_KEY>
EOF
```

TEXT

```
cat << EOF | sensuctl create
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "sensu-ec2-handler"
  },
  "spec": {
    "type": "pipe",
    "runtime_assets": [
      "sensu/sensu-ec2-handler"
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "command": "sensu-ec2-handler --aws-region <AWS_REGION> --aws-instance-id-label
AWS_INSTANCE_ID_LABEL --aws-allowed-instance-states pending,running --sensu-api-url
<http://localhost:8080">,
    "secrets": [
      {
        "name": "AWS_ACCESS_KEY_ID",
        "secret": "<YOUR_AWS_ACCESS_KEY_ID>"
      },
      {
        "name": "AWS_SECRET_KEY",
        "secret": "<YOUR_AWS_SECRET_KEY>"
      },
      {
        "name": "SENSU_API_KEY",
        "secret": "<YOUR_SENSU_API_KEY>"
      }
    ]
  }
}
EOF
```

The handler will use the provided AWS credentials to check the specified EC2 instance. If the instance's status is not "pending" or "running," the handler will use the `ec2-service` user's API key to remove the corresponding entity.

> **NOTE**: *Instead of directly referencing your* `AWS_ACCESS_KEY_ID` *,* `AWS_SECRET_KEY` *, and* `SENSU_API_KEY` *as shown in the* `sensu-ec2-handler` *example handler definition above, use* secrets management *to configure these values as environment variables.*

Best practices for limited service accounts

Follow these best practices for creating and managing limited service accounts:

- Use unique and specific names for limited service accounts. Names should identify the accounts as limited service accounts as well as the associated services.

- Restrict limited service account access to only the namespaces and role permissions they need to operate properly. Adjust namespaces and permissions if needed by updating the role or cluster role that is tied to the service account.

- Promptly delete unused limited service accounts to make sure they do not become security risks.

Active Directory (AD) reference

> **COMMERCIAL FEATURE**: Access active directory (AD) authentication for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read [Get started with commercial features](#).

Sensu requires username and password authentication to access the [web UI](#), [API](#), and [sensuctl](#) command line tool.

In addition to the [built-in basic authentication provider](#), Sensu offers [commercial support](#) for using Microsoft Active Directory (AD) for single sign-on (SSO) authentication. The AD authentication provider is based on the [LDAP authentication provider](#).

To use AD authentication for Azure, follow Microsoft's tutorial to [set up secure LDAP in your Azure account](#) and create the host and certificates you need.

For general information about configuring authentication providers, read [Configure single sign-on (SSO) authentication](#).

AD configuration examples

Example AD configuration: Minimum required attributes

YML

```
---
type: ad
api_version: authentication/v2
metadata:
  name: activedirectory
spec:
  servers:
  - group_search:
      base_dn: dc=acme,dc=org
    host: 127.0.0.1
    user_search:
```

```
    base_dn: dc=acme,dc=org
```

JSON

```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:
  name: activedirectory
spec:
  allowed_groups: []
  groups_prefix: ad
  servers:
  - binding:
      password: YOUR_PASSWORD
```

```
        user_dn: cn=binder,cn=users,dc=acme,dc=org
      client_cert_file: /path/to/ssl/cert.pem
      client_key_file: /path/to/ssl/key.pem
      default_upn_domain: example.org
      include_nested_groups: true
      group_search:
        attribute: member
        base_dn: dc=acme,dc=org
        name_attribute: cn
        object_class: group
      host: 127.0.0.1
      insecure: false
      port: 636
      security: tls
      trusted_ca_file: /path/to/trusted-certificate-authorities.pem
      user_search:
        attribute: sAMAccountName
        base_dn: dc=acme,dc=org
        name_attribute: displayName
        object_class: person
    username_prefix: ad
```

JSON

```json
{
  "type": "ad",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "default_upn_domain": "example.org",
        "include_nested_groups": true,
        "binding": {
          "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
```

```
        },
        "group_search": {
            "base_dn": "dc=acme,dc=org",
            "attribute": "member",
            "name_attribute": "cn",
            "object_class": "group"
        },
        "user_search": {
            "base_dn": "dc=acme,dc=org",
            "attribute": "sAMAccountName",
            "name_attribute": "displayName",
            "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ad",
    "username_prefix": "ad"
  },
  "metadata": {
    "name": "activedirectory"
  }
}
```

Example AD configuration: Use `memberOf` attribute instead of `group_search`

AD automatically returns a `memberOf` attribute in users' accounts. The `memberOf` attribute contains the user's group membership, which effectively removes the requirement to look up the user's groups.

To use the `memberOf` attribute in your AD implementation, remove the `group_search` object from your AD config:

YML

```yml
---
type: ad
api_version: authentication/v2
metadata:
  name: activedirectory
spec:
  servers:
```

```
    host: 127.0.0.1
    user_search:
      base_dn: dc=acme,dc=org
```

JSON

```json
{
  "type": "ad",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "activedirectory"
  }
}
```

After you configure AD to use the `memberOf` attribute, the `debug` log level will include the following log entries:

```
{"component":"authentication/v2","level":"debug","msg":"using the \"memberOf\"
attribute to determine the group membership of user \"user1\"","time":"2020-06-
25T14:10:58-04:00"}
{"component":"authentication/v2","level":"debug","msg":"found 1 LDAP group(s):
[\"sensu\"]","time":"2020-06-25T14:10:58-04:00"}
```

AD specification

AD top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. For AD definitions, the `type` should always be `ad`.
required	true
type	String

example

YML

```
type: ad
```

JSON

```
{
  "type": "ad"
}
```

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For AD definitions, the `api_version` should always be `authentication/v2`.
required	true
type	String

example

YML

```
api_version: authentication/v2
```

JSON

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

```
        }
      - host: 127.0.0.1
```

metadata

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

YML

example

```yaml
metadata:
  name: activedirectory
```

JSON

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

spec

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

YML

example

```yaml
spec:
  servers:
  - host: 127.0.0.1
```

```
      port: 636
      insecure: false
      security: tls
      trusted_ca_file: "/path/to/trusted-certificate-
authorities.pem"
      client_cert_file: "/path/to/ssl/cert.pem"
      client_key_file: "/path/to/ssl/key.pem"
      default_upn_domain: example.org
      include_nested_groups: true
      binding:
        user_dn: cn=binder,cn=users,dc=acme,dc=org
        password: YOUR_PASSWORD
      group_search:
        base_dn: dc=acme,dc=org
        attribute: member
        name_attribute: cn
        object_class: group
      user_search:
        base_dn: dc=acme,dc=org
        attribute: sAMAccountName
        name_attribute: displayName
        object_class: person
    allowed_groups: []
    groups_prefix: ad
    username_prefix: ad
```

JSON

```
{
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "default_upn_domain": "example.org",
        "include_nested_groups": true,
```

```
        "binding": {
          "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "member",
          "name_attribute": "cn",
          "object_class": "group"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "sAMAccountName",
          "name_attribute": "displayName",
          "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ad",
    "username_prefix": "ad"
  }
}
```

AD spec attributes

servers	
description	The list of AD servers to use. During the authentication process, Sensu attempts to authenticate against each AD server in sequence until authentication is successful or there are no more servers to try.
required	true
type	Array
	YML
example	

```
servers:
```

```yaml
  - host: 127.0.0.1
    port: 636
    insecure: false
    security: tls
    trusted_ca_file: "/path/to/trusted-certificate-
authorities.pem"
    client_cert_file: "/path/to/ssl/cert.pem"
    client_key_file: "/path/to/ssl/key.pem"
    default_upn_domain: example.org
    include_nested_groups: true
    binding:
      user_dn: cn=binder,cn=users,dc=acme,dc=org
      password: YOUR_PASSWORD
    group_search:
      base_dn: dc=acme,dc=org
      attribute: member
      name_attribute: cn
      object_class: group
    user_search:
      base_dn: dc=acme,dc=org
      attribute: sAMAccountName
      name_attribute: displayName
      object_class: person
```

JSON

```json
{
  "servers": [
    {
      "host": "127.0.0.1",
      "port": 636,
      "insecure": false,
      "security": "tls",
      "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem",
      "client_cert_file": "/path/to/ssl/cert.pem",
      "client_key_file": "/path/to/ssl/key.pem",
      "default_upn_domain": "example.org",
      "include_nested_groups": true,
      "binding": {
        "user_dn": "cn=binder,cn=users,dc=acme,dc=org",
```

```
            "password": "YOUR_PASSWORD"
          },
          "group_search": {
            "base_dn": "dc=acme,dc=org",
            "attribute": "member",
            "name_attribute": "cn",
            "object_class": "group"
          },
          "user_search": {
            "base_dn": "dc=acme,dc=org",
            "attribute": "sAMAccountName",
            "name_attribute": "displayName",
            "object_class": "person"
          }
        }
      ]
    }
```

allowed_groups	
description	An array of allowed AD group strings to include in the tokenized identity claim. Use to specify which groups to encode in the authentication provider's JSON Web Token (JWT) when the authenticated AD user is a member of many groups and the tokenized identity claim would be too large for correct web client operation.
required	false
type	Array
	YML
example	allowed_groups: - sensu-viewers - sensu-operators **JSON** { "allowed groups": [

```
      "sensu-viewers",
      "sensu-operators"
    ]
  }
```

groups_prefix

description	The prefix added to all AD groups. Sensu appends the groups_prefix with a colon. For example, for the groups_prefix `ad` and the group `dev`, the resulting group name in Sensu is `ad:dev`. Use the groups_prefix when integrating AD groups with Sensu RBAC <u>role bindings and cluster role bindings</u>.
required	false
type	String

YML

| example | ```
groups_prefix: ad
``` |

JSON

```
{
  "groups_prefix": "ad"
}
```

username_prefix

description	The prefix added to all AD usernames. Sensu appends the username_prefix with a colon. For example, for the username_prefix `ad` and the user `alice`, the resulting username in Sensu is `ad:alice`. Use the username_prefix when integrating AD users with Sensu RBAC <u>role bindings and cluster role bindings</u>. Users *do not* need to provide the username_prefix when logging in to Sensu.

required	false
type	String

example

YML

```
username_prefix: ad
```

JSON

```json
{
  "username_prefix": "ad"
}
```

AD server attributes

host	
description	AD server IP address or FQDN.
required	true
type	String

example

YML

```
host: 127.0.0.1
```

JSON

```json
{
  "host": "127.0.0.1"
}
```

port	

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

```yml
  port: 636
```

JSON

```json
{
  "port": 636
}
```

insecure

description	Skips SSL certificate verification when set to `true`. **WARNING**: *Do not use an insecure connection in production environments.*
required	false
type	Boolean
default	`false`
example	**YML**

```yml
  insecure: false
```

JSON

```json
{
```

```
    "insecure": false
  }
```

security

description	Determines the encryption type to be used for the connection to the AD server: `insecure` (unencrypted connection; not recommended for production), `tls` (secure encrypted connection), or `starttls` (unencrypted connection upgrades to a secure connection).
type	String
default	`"tls"`

YML

example

```
  security: tls
```

JSON

```
  {
    "security": "tls"
  }
```

trusted_ca_file

description	Path to an alternative CA bundle file in PEM format to be used instead of the system's default bundle. This CA bundle is used to verify the server's certificate.
required	false
type	String

YML

example

```
  trusted_ca_file: /path/to/trusted-certificate-authorities.pem
```

JSON

```json
{
  "trusted_ca_file": "/path/to/trusted-certificate-
authorities.pem"
}
```

client_cert_file

description	Path to the certificate that should be sent to the server if requested.
required	false
type	String

YML

example

```yml
client_cert_file: /path/to/ssl/cert.pem
```

JSON

```json
{
  "client_cert_file": "/path/to/ssl/cert.pem"
}
```

client_key_file

description	Path to the key file associated with the `client_cert_file`.
required	false
type	String

YML

example

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

JSON

```json
{
  "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

YML

example	

```yaml
binding:
  user_dn: cn=binder,cn=users,dc=acme,dc=org
  password: YOUR_PASSWORD
```

JSON

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

group_search

description	Search configuration for groups. Review the group search attributes for more information. Remove the `group_search` object from your configuration to use the `memberOf` attribute instead.
required	false
type	Map

YML

example

```yaml
group_search:
  base_dn: dc=acme,dc=org
  attribute: member
  name_attribute: cn
  object_class: group
```

JSON

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

user_search

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

YML

example

```yaml
user_search:
  base_dn: dc=acme,dc=org
```

```
    attribute: sAMAccountName
    name_attribute: displayName
    object_class: person
```

JSON

```json
{
  "user_search": {
    "base_dn": "dc=acme,dc=org",
    "attribute": "sAMAccountName",
    "name_attribute": "displayName",
    "object_class": "person"
  }
}
```

default_upn_domain	
description	Enables UPN authentication when set. The default UPN suffix that will be appended to the username when a domain is not specified during login (for example, `user` becomes `user@defaultdomain.xyz`).
	WARNING: When using UPN authentication, users must re-authenticate to apply any changes to group membership on the AD server since their last authentication. For example, if you remove a user from a group with administrator permissions for the current session (such as a terminated employee), Sensu will not apply the change until the user logs out and tries to start a new session. Likewise, under UPN, users cannot be forced to log out of Sensu. To apply group membership updates without re-authentication, specify a binding account or enable anonymous binding.
required	false
type	String
	YML
example	```default_upn_domain: example.org```

JSON

```json
{
  "default_upn_domain": "example.org"
}
```

include_nested_groups

description	If `true` , the group search includes any nested groups a user is a member of. If `false` , the group search includes only the top-level groups a user is a member of.
required	false
type	Boolean

example	**YML**
	
	`include_nested_groups: true`
	
	JSON
	
	``json
	{
	  "include_nested_groups": true
	}
	``

AD binding attributes

user_dn

description	The AD account that performs user and group lookups. We recommend using a read-only account. Use the distinguished name (DN) format, such as `cn=binder,cn=users,dc=domain,dc=tld` . If your sever

supports anonymous binding, you can omit this attribute to query the directory without credentials.

required	false
type	String

YML

example

```yaml
user_dn: cn=binder,cn=users,dc=acme,dc=org
```

JSON

```json
{
  "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

YML

example

```yaml
password: YOUR_PASSWORD
```

JSON

```json
{
  "password": "YOUR_PASSWORD"
}
```

AD group search attributes

base_dn	
description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String
example	**YML**

```yml
base_dn: dc=acme,dc=org
```

JSON

```json
{
  "base_dn": "dc=acme,dc=org"
}
```

attribute	
description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false
type	String
default	`"member"`
example	**YML**

```yml
attribute: member
```

JSON

```json
{
```

```
    "attribute": "member"
  }
```

name_attribute

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

YML

example

```
  name_attribute: cn
```

JSON

```
  {
    "name_attribute": "cn"
  }
```

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`"group"`

YML

example

```
  object_class: group
```

JSON

```json
{
  "object_class": "group"
}
```

AD user search attributes

base_dn	
description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String

example	**YML**
	
	`base_dn: dc=acme,dc=org`
	
	JSON
	
	``json
	{
	  "base_dn": "dc=acme,dc=org"
	}
	``

attribute	
description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false

type	String

default	`"sAMAccountName"`

YML

example	

```yaml
  attribute: sAMAccountName
```

JSON

```json
  {
    "attribute": "sAMAccountName"
  }
```

name_attribute

description	Represents the attribute to use as the entry name.

required	false

type	String

default	`"displayName"`

YML

example	

```yaml
  name_attribute: displayName
```

JSON

```json
  {
    "name_attribute": "displayName"
  }
```

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`"person"`

YML

```yaml
object_class: person
```

JSON

```json
{
  "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

YML

```yaml
name: activedirectory
```

JSON

```json
{
  "name": "activedirectory"
```

```
    }
```

AD troubleshooting

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

Lightweight Directory Access Protocol (LDAP) reference

> **COMMERCIAL FEATURE**: Access Lightweight Directory Access Protocol (LDAP) authentication for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read [Get started with commercial features](#).

Sensu requires username and password authentication to access the [web UI](#), [API](#), and [sensuctl](#) command line tool.

In addition to the [built-in basic authentication provider](#), Sensu offers [commercial support](#) for a standards-compliant Lightweight Directory Access Protocol (LDAP) tool for single sign-on (SSO) authentication. The Sensu LDAP authentication provider is tested with [OpenLDAP](#). If you're using AD, head to the [AD section](#).

For general information about configuring authentication providers, read [Configure single sign-on (SSO) authentication](#).

LDAP configuration examples

Example LDAP configuration: Minimum required attributes

YML

```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

```yaml
---
type: ldap
api_version: authentication/v2
metadata:
  name: openldap
spec:
  allowed_groups: []
  groups_prefix: ldap
  servers:
  - binding:
      password: YOUR_PASSWORD
```

```yaml
      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",
```

```json
        "name_attribute": "cn",
        "object_class": "groupOfNames"
      },
      "user_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "uid",
        "name_attribute": "cn",
        "object_class": "person"
      }
    }
  ],
  "allowed_groups": [],
  "groups_prefix": "ldap",
  "username_prefix": "ldap"
  },
  "metadata": {
    "name": "openldap"
  }
}
```

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	**YML**
	
	``yml
	type: ldap
	``
	
	JSON

```
{
  "type": "ldap"
}
```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For LDAP definitions, the `api_version` should always be `authentication/v2` .
required	true
type	String

example

YML

```
api_version: authentication/v2
```

JSON

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

metadata

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

example

YML

```
metadata:
  name: openldap
```

JSON

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

spec

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

YML

example	

```yaml
spec:
  servers:
  - host: 127.0.0.1
    port: 636
    insecure: false
    security: tls
    trusted_ca_file: "/path/to/trusted-certificate-
authorities.pem"
    client_cert_file: "/path/to/ssl/cert.pem"
    client_key_file: "/path/to/ssl/key.pem"
    binding:
      user_dn: cn=binder,dc=acme,dc=org
      password: YOUR_PASSWORD
    group_search:
      base_dn: dc=acme,dc=org
      attribute: member
      name_attribute: cn
      object_class: groupOfNames
    user_search:
      base_dn: dc=acme,dc=org
```

```
      attribute: uid
      name_attribute: cn
      object_class: person
  allowed_groups: []
  groups_prefix: ldap
  username_prefix: ldap
```

JSON

```json
{
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "port": 636,
        "insecure": false,
        "security": "tls",
        "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem",
        "client_cert_file": "/path/to/ssl/cert.pem",
        "client_key_file": "/path/to/ssl/key.pem",
        "binding": {
          "user_dn": "cn=binder,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "member",
          "name_attribute": "cn",
          "object_class": "groupOfNames"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org",
          "attribute": "uid",
          "name_attribute": "cn",
          "object_class": "person"
        }
      }
    ],
    "allowed_groups": [],
    "groups_prefix": "ldap",
    "username_prefix": "ldap"
```

```
    }
  }
```

LDAP spec attributes

servers	
description	The list of LDAP servers to use. During the authentication process, Sensu attempts to authenticate against each LDAP server in sequence until authentication is successful or there are no more servers to try.
required	true
type	Array

YML

example	

```yml
servers:
- host: 127.0.0.1
  port: 636
  insecure: false
  security: tls
  trusted_ca_file: "/path/to/trusted-certificate-authorities.pem"
  client_cert_file: "/path/to/ssl/cert.pem"
  client_key_file: "/path/to/ssl/key.pem"
  binding:
    user_dn: cn=binder,dc=acme,dc=org
    password: YOUR_PASSWORD
  group_search:
    base_dn: dc=acme,dc=org
    attribute: member
    name_attribute: cn
    object_class: groupOfNames
  user_search:
    base_dn: dc=acme,dc=org
    attribute: uid
    name_attribute: cn
    object_class: person
```

JSON

```json
{
  "servers": [
    {
      "host": "127.0.0.1",
      "port": 636,
      "insecure": false,
      "security": "tls",
      "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem",
      "client_cert_file": "/path/to/ssl/cert.pem",
      "client_key_file": "/path/to/ssl/key.pem",
      "binding": {
        "user_dn": "cn=binder,dc=acme,dc=org",
        "password": "YOUR_PASSWORD"
      },
      "group_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "member",
        "name_attribute": "cn",
        "object_class": "groupOfNames"
      },
      "user_search": {
        "base_dn": "dc=acme,dc=org",
        "attribute": "uid",
        "name_attribute": "cn",
        "object_class": "person"
      }
    }
  ]
}
```

allowed_groups

description	An array of allowed LDAP group strings to include in the tokenized identity claim. Use to specify which groups to encode in the

authentication provider's JSON Web Token (JWT) when the authenticated LDAP user is a member of many groups and the tokenized identity claim would be too large for correct web client operation.

required	false
type	Array of strings

example

YML

```
allowed_groups:
- sensu-viewers
- sensu-operators
```

JSON

```
{
  "allowed_groups": [
    "sensu-viewers",
    "sensu-operators"
  ]
}
```

groups_prefix

description	The prefix added to all LDAP groups. Sensu appends the groups_prefix with a colon. For example, for the groups_prefix `ldap` and the group `dev`, the resulting group name in Sensu is `ldap:dev`. Use the groups_prefix when integrating LDAP groups with Sensu RBAC <u>role bindings and cluster role bindings</u>.
required	false
type	String

example

YML

```
groups_prefix: ldap
```

JSON

```json
{
  "groups_prefix": "ldap"
}
```

username_prefix

description	The prefix added to all LDAP usernames. Sensu appends the username_prefix with a colon. For example, for the username_prefix `ldap` and the user `alice`, the resulting username in Sensu is `ldap:alice`. Use the username_prefix when integrating LDAP users with Sensu RBAC role bindings and cluster role bindings. Users *do not* need to provide the username_prefix when logging in to Sensu.
required	false
type	String

example	**YML**
	
	``yml
	username_prefix: ldap
	``
	
	JSON
	
	``json
	{
	  "username_prefix": "ldap"
	}
	``

LDAP server attributes

host

description	LDAP server IP address or FQDN.
required	true

type	String

YML

```yaml
host: 127.0.0.1
```

JSON

```json
{
  "host": "127.0.0.1"
}
```

port

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

YML

```yaml
port: 636
```

JSON

```json
{
  "port": 636
}
```

insecure

description	Skips SSL certificate verification when set to `true`.

required	false
type	Boolean
default	`false`

example	**YML**

```yml
insecure: false
```

JSON

```json
{
  "insecure": false
}
```

security

description	Determines the encryption type to be used for the connection to the LDAP server: `insecure` (unencrypted connection; not recommended for production), `tls` (secure encrypted connection), or `starttls` (unencrypted connection upgrades to a secure connection).
type	String
default	`"tls"`

example	**YML**

```yml
security: tls
```

JSON

```json
{
  "security": "tls"
}
```

trusted_ca_file

description	Path to an alternative CA bundle file in PEM format to be used instead of the system's default bundle. This CA bundle is used to verify the server's certificate.
required	false
type	String
| example | **YML**<br><br>```yml
trusted_ca_file: /path/to/trusted-certificate-authorities.pem
```<br><br>**JSON**<br><br>```json
{
  "trusted_ca_file": "/path/to/trusted-certificate-authorities.pem"
}
``` |

client_cert_file

description	Path to the certificate that should be sent to the server if requested.
required	false
type	String
| example | **YML**<br><br>```yml
client_cert_file: /path/to/ssl/cert.pem
```<br><br>**JSON**<br><br>```json
{
``` |

```
      "client_cert_file": "/path/to/ssl/cert.pem"

    }
```

client_key_file

description	Path to the key file associated with the `client_cert_file`.
required	false
type	String

example

YML

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

JSON

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

binding

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

example

YML

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

JSON

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

group_search

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

YML

example	

```yaml
group_search:
  base_dn: dc=acme,dc=org
  attribute: member
  name_attribute: cn
  object_class: groupOfNames
```

JSON

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

user_search	
description	Search configuration for users. Review the <u>user search attributes</u> for more information.
required	true
type	Map

example

YML

```yml
user_search:
  base_dn: dc=acme,dc=org
  attribute: uid
  name_attribute: cn
  object_class: person
```

JSON

```json
{
  "user_search": {
    "base_dn": "dc=acme,dc=org",
    "attribute": "uid",
    "name_attribute": "cn",
    "object_class": "person"
  }
}
```

LDAP binding attributes

user_dn	
description	The LDAP account that performs user and group lookups. We recommend using a read-only account. Use the distinguished name (DN) format, such as `cn=binder,cn=users,dc=domain,dc=tld`. If your sever supports anonymous binding, you can omit this attribute to query the directory without credentials.

required	false
type	String

example	**YML**

```yaml
  user_dn: cn=binder,dc=acme,dc=org
```

JSON

```json
{
  "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	**YML**

```yaml
  password: YOUR_PASSWORD
```

JSON

```json
{
  "password": "YOUR_PASSWORD"
}
```

LDAP group search attributes

base_dn

description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String
example	**YML**

```yml
base_dn: dc=acme,dc=org
```

JSON

```json
{
  "base_dn": "dc=acme,dc=org"
}
```

attribute

description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false
type	String
default	`"member"`
example	**YML**

```yml
attribute: member
```

JSON

```json
{
  "attribute": "member"
}
```

name_attribute

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

```yml
name_attribute: cn
```

JSON

```json
{
  "name_attribute": "cn"
}
```

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"`.
required	false
type	String
default	`"groupOfNames"`
example	**YML**

```yml
object_class: groupOfNames
```

JSON

```json
{
  "object_class": "groupOfNames"
}
```

LDAP user search attributes

base_dn	
description	Tells Sensu which part of the directory tree to search. For example, `dc=acme,dc=org` searches within the `acme.org` directory.
required	true
type	String

example	**YML**
	
	base_dn: dc=acme,dc=org
	
	JSON
	
	``json
	{
	  "base_dn": "dc=acme,dc=org"
	}
	``

attribute	
description	Used for comparing result entries. Combined with other filters as `"(<Attribute>=<value>)"`.
required	false
type	String

default	`"uid"`

YML

example	

```
attribute: uid
```

JSON

```
{
  "attribute": "uid"
}
```

name_attribute

description	Represents the attribute to use as the entry name

required	false

type	String

default	`"cn"`

YML

example	

```
name_attribute: cn
```

JSON

```
{
  "name_attribute": "cn"
}
```

object_class

description	Identifies the class of objects returned in the search result. Combined with other filters as `"(objectClass=<ObjectClass>)"` .

required	false
type	String
default	`"person"`

example

YML

```
object_class: person
```

JSON

```
{
  "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

YML

```
name: openldap
```

JSON

```
{
  "name": "openldap"
}
```

LDAP troubleshooting

To troubleshoot any issue with LDAP authentication, start by [increasing the log verbosity][19] 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 [...]
```

OpenID Connect 1.0 protocol (OIDC) reference

> **COMMERCIAL FEATURE**: Access OpenID Connect 1.0 protocol (OIDC) authentication for single sign-on (SSO) in the packaged Sensu Go distribution. For more information, read [Get started with commercial features](#).

Sensu requires username and password authentication to access the [web UI](#), [API](#), and [sensuctl](#) command line tool.

In addition to the [built-in basic authentication provider](#), Sensu offers [commercial support](#) for single sign-on (SSO) authentication using the OpenID Connect 1.0 protocol (OIDC) on top of the OAuth 2.0 protocol. The Sensu OIDC provider is tested with [Okta](#) and [PingFederate](#).

For general information about configuring authentication providers, read [Configure single sign-on (SSO) authentication](#).

> **WARNING**: Defining multiple OIDC providers can lead to inconsistent authentication behavior. Use `sensuctl auth list` to verify that only one authentication provider of type `OIDC` is defined. If more than one OIDC auth provider configuration is listed, use `sensuctl auth delete $NAME` to remove the extra OIDC configurations by name.

OIDC configuration examples

YML

```yml
---
type: oidc
api_version: authentication/v2
metadata:
  name: oidc_name
spec:
  additional_scopes:
  - groups
  - email
```

```
client_id: a8e43af034e7f2608780

client_secret: b63968394be6ed2edb61c93847ee792f31bf6216

disable_offline_access: false

redirect_uri: http://127.0.0.1:8080/api/enterprise/authentication/v2/oidc/callback

server: https://oidc.example.com:9031

groups_claim: groups

groups_prefix: 'oidc:'

username_claim: email

username_prefix: 'oidc:'
```

JSON

```json
{
  "type": "oidc",
  "api_version": "authentication/v2",
  "metadata": {
    "name": "oidc_name"
  },
  "spec": {
    "additional_scopes": [
      "groups",
      "email"
    ],
    "client_id": "a8e43af034e7f2608780",
    "client_secret": "b63968394be6ed2edb61c93847ee792f31bf6216",
    "disable_offline_access": false,
    "redirect_uri": "http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/oidc/callback",
    "server": "https://oidc.example.com:9031",
    "groups_claim": "groups",
    "groups_prefix": "oidc:",
    "username_claim": "email",
    "username_prefix": "oidc:"
  }
}
```

OIDC specification

OIDC top-level attributes

type

description	Top-level attribute that specifies the `sensuctl create` resource type. For OIDC configuration, the `type` should always be `oidc`.
required	true
type	String

example	
	YML

```yml
type: oidc
```

JSON

```json
{
  "type": "oidc"
}
```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For OIDC configuration, the `api_version` should always be `authentication/v2`.
required	true
type	String

example	
	YML

```yml
api_version: authentication/v2
```

JSON

```json
{
  "api_version": "authentication/v2"
```

```
  }
```

metadata

description	Top-level collection of metadata about the OIDC configuration. The `metadata` map is always at the top level of the OIDC definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope.
required	true
type	Map of key-value pairs

example

YML

```
metadata:
  name: oidc_name
```

JSON

```
{
  "metadata": {
    "name": "oidc_name"
  }
}
```

spec

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

example

YML

```
spec:
```

```yaml
  additional_scopes:
  - groups
  - email
  client_id: a8e43af034e7f2608780
  client_secret: b63968394be6ed2edb61c93847ee792f31bf6216
  disable_offline_access: false
  redirect_uri: http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/o
idc/callback
  server: https://oidc.example.com:9031
  groups_claim: groups
  groups_prefix: 'oidc:'
  username_claim: email
  username_prefix: 'oidc:'
```

JSON

```json
{
  "spec": {
    "additional_scopes": [
      "groups",
      "email"
    ],
    "client_id": "a8e43af034e7f2608780",
    "client_secret":
"b63968394be6ed2edb61c93847ee792f31bf6216",
    "disable_offline_access": false,
    "redirect_uri": "http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/o
idc/callback",
    "server": "https://oidc.example.com:9031",
    "groups_claim": "groups",
    "groups_prefix": "oidc:",
    "username_claim": "email",
    "username_prefix": "oidc:"
  }
}
```

OIDC metadata attribute

name	
description	A unique string used to identify the OIDC configuration. The `metadata.name` cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`).
required	true
type	String

example	**YML**
	
	```yml\nname: oidc_name\n```
	
	JSON
	
	```json\n{\n "name": "oidc_name"\n}\n```

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	**YML**

```yaml
additional_scopes:
- groups
- email
- username
```

JSON

```json
{
  "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

YML

```yaml
client_id: 1c9ae3e6f3cc79c9f1786fcb22692d1f
```

JSON

```json
{
  "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

YML

```yml
client_secret: a0f2a3c1dcd5b1cac71bf0c03f2ff1bd
```

JSON

```json
{
  "client_secret": "a0f2a3c1dcd5b1cac71bf0c03f2ff1bd"
}
```

disable_offline_access

description	If `true`, the OIDC provider cannot include the `offline_access` scope in the authentication request. Otherwise, `false`. We recommend setting `disable_offline_access` to `false`. If set to `true`, OIDC providers cannot return a refresh token that allows users to refresh their access tokens, and users will be logged out after 5 minutes.
required	true
default	false
type	Boolean

YML

example	
| | ```
disable_offline_access: false
``` |

JSON

```json
{
  "disable_offline_access": false
}
```

redirect_uri

description	Redirect URL to provide to the OIDC provider. Requires `/api/enterprise/authentication/v2/oidc/callback`
	NOTE: Only required for certain OIDC providers, such as Okta.

required	false
type	String
	YML
| example | ```
redirect_uri: http://sensu-
backend.example.com:8080/api/enterprise/authentication/v2/o
idc/callback
``` |

JSON

```json
{
  "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

YML

```
server: https://sensu.oidc.provider.example.com
```

JSON

```
{
  "server": "https://sensu.oidc.provider.example.com"
}
```

groups_claim	
description	The claim to use to form the associated RBAC groups.
	NOTE: *The value held by the claim must be an array of strings.*
required	false
type	String

YML

```
groups_claim: groups
```

JSON

```
{
  "groups_claim": "groups"
}
```

groups_prefix

description	The prefix added to all OIDC groups. Sensu appends the groups_prefix with a colon. For example, for the groups_prefix `okta` and the group `dev` , the resulting group name in Sensu is `okta:dev` . Use the groups_prefix when integrating OIDC groups with Sensu RBAC role bindings and cluster role bindings.
required	false
type	String

example

YML

```
groups_prefix: okta
```

JSON

```
{
  "groups_prefix": "okta"
}
```

username_claim

description	The claim to use to form the final RBAC user name.
required	false
type	String

example

YML

```
username_claim: person
```

JSON

```json
{
  "username_claim": "person"
}
```

username_prefix	
description	The prefix added to all OIDC usernames. Sensu appends the username_prefix with a colon. For example, for the username_prefix `okta` and the user `alice`, the resulting username in Sensu is `okta:alice`. Use the username_prefix when integrating OIDC users with Sensu RBAC role bindings and cluster role bindings. Users *do not* need to provide the username_prefix when logging in to Sensu.
required	false
type	String

example	**YML**
	
	``yml
	username_prefix: okta
	``
	
	JSON
	
	``json
	{
	  "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 with a valid commercial license

Create an Okta application

> **NOTE***: These instructions are based on the Okta Classic UI. The steps may be different if you are using the Okta Developer Console.*

1. In the Okta Administrator Dashboard, start the wizard:
   select `Applications` > `Add Application` > `Create New App` .
2. In the *Platform* dropdown, select `Web` .
3. In the *Sign on method* section, select `OpenID Connect` .
4. Click **Create**.
5. In the *Create OpenID Connect Integration* window:

   ¬ *GENERAL SETTINGS* section: in the *Application name* field, enter the app name. You can also upload a logo in the if desired.

   ¬ *CONFIGURE OPENID CONNECT* section: in the *Login redirect URIs* field, enter `<api_url>/api/enterprise/authentication/v2/oidc/callback` .Replace `<api_url>` with your API URL, including the API port 8080.

6. Click **Save**.
7. Select the *General* tab and click **Edit**.
8. In the *Allowed grant types* section, click to select the box next to **Refresh Token** .
9. Click **Save**.
10. Select the *Sign On* tab.
11. In the *OpenID Connect ID Token* section, click **Edit**.
12. In the *Groups claim filter* section:

    ¬ In the first field, enter `groups`

    ¬ In the dropdown menu, select `matches regex`

    ¬

⌐ In the second field, enter `.*`

13. Click **Save**.
14. (Optional) Select the *Assignments* tab to assign people and groups to your app.

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 3 of Create an Okta application:

    ⌐ `"redirect_uri": "`
       `<api_url>/api/enterprise/authentication/v2/oidc/callback"`

4. Configure authorization for your OIDC users and groups by creating roles (or cluster roles) and role bindings (or cluster role bindings) that map to the user and group names.

> **NOTE**: If you do not configure authorization, users will be able to log in with OIDC but will have no permissions by default.

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:

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

API keys reference

API keys are long-lived authentication tokens that make it more convenient for Sensu plugins and other Sensu-adjacent applications to authenticate with the Sensu API. Unlike <u>authentication tokens</u>, API keys are persistent and do not need to be refreshed every 15 minutes.

The Sensu backend generates API keys, and you can provide them to applications that want to interact with the Sensu API.

Use the <u>APIKey API</u> to create, retrieve, and delete API keys.

API key example

This example shows an `APIKey` resource definition:

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"
```

```
    }
  }
```

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 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	YML
	```type: APIKey```

JSON

```
{
  "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	**YML**
	
	`api_version: core/v2`
	
	JSON
	
	`{`
	  `"api_version": "core/v2"`
	`}`

metadata

description	Top-level collection of metadata about the API key, including `name` and `created_by` . The `metadata` map is always at the top level of the API key definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope.
required	true
type	Map of key-value pairs

example	YML

```yaml
metadata:
  name: 19803eb8-36a6-4203-a225-28ec4e9f4444
  created_by: admin
```

JSON

```json
{
  "metadata": {
    "name": "19803eb8-36a6-4203-a225-28ec4e9f4444",
    "created_by": "admin"
  }
}
```

spec

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

YML

example	

```yaml
spec:
  created_at: 1570732266
  username: admin
```

JSON

```json
{
  "spec": {
    "created_at": 1570732266,
    "username": "admin"
  }
}
```

Metadata attributes

name	
description	Unique string used to identify the API key. Sensu randomly generates a universally unique identifier (UUID) for the `name` value — users cannot provide a name for an API key.
required	true
type	String
example	YML

```yml
name: 19803eb8-36a6-4203-a225-28ec4e9f4444
```

JSON

```json
{
  "name": "19803eb8-36a6-4203-a225-28ec4e9f4444"
}
```

created_by	
description	Username of the Sensu user who created the API key or last updated the API key. Sensu automatically populates the `created_by` field when the API key is created or updated.
required	false
type	String
example	YML

```yml
created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

Spec attributes

username	
description	User associated with the API key.
required	true
type	Array

YML

```yaml
  username: admin
```

JSON

```json
{
  "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

YML

example	

```
  created_at: 1234567890
```

JSON

```json
{
  "created_at": 1234567890
}
```

Namespaces reference

Namespaces partition resources within Sensu. Sensu entities, checks, handlers, and other namespaced resources belong to a single namespace.

Namespaces help teams use different resources (like entities, checks, and handlers) within Sensu and impose their own controls on those resources. A Sensu instance can have multiple namespaces, each with their own set of managed resources. Resource names must be unique within a namespace but do not need to be unique across namespaces.

Namespace configuration applies to sensuctl, the API, and the web UI. To create and manage namespaces, configure sensuctl as the default `admin` user or create a cluster role with `namespaces` permissions.

Namespace example

This example shows the resource definition for a `production` namespace. You can use this example with `sensuctl create` to create a `production` namespace in your Sensu deployment:

YML

```yaml
---
type: Namespace
api_version: core/v2
metadata: {}
spec:
  name: production
```

JSON

```json
{
  "type": "Namespace",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "name": "production"
  }
}
```

```
}
```

Best practices for namespaces

Use namespaces for isolation, not organization

Use namespaces to enforce full isolation of different groups of resources, not to organize resources. An agent cannot belong to two namespaces or execute checks in two different namespaces. To organize resources, use labels rather than multiple namespaces.

Default namespaces

Every Sensu backend includes a `default` namespace. All resources created without a specified namespace are created within the `default` namespace.

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

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

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>
```

Namespaces must be empty before you can delete them. If the response to `sensuctl namespace delete` is `Error: resource is invalid: namespace is not empty`, the namespace may still contain events or other resources. To remove all resources and events so that you can delete a namespace, run this command (replace `<namespace-name>` with the namespace you want to empty):

```
sensuctl dump entities,events,assets,checks,filters,handlers,secrets/v1.Secret --
namespace <namespace-name> | sensuctl delete
```

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

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-cpu",
    "namespace": "production"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu.sh -w 75 -c 90",
    "handlers": ["slack"],
    "interval": 30,
    "subscriptions": ["system"],
    "timeout": 0,
    "ttl": 0
  }
}
```

Read the reference docs for the corresponding resource type to create resource definitions.

Namespace specification

Top-level attributes

type	
description	Top-level attribute that specifies the resource type. Namespaces should always be type `Namespace`.
required	true
type	String
| example | **YML**<br><br>```yaml
type: Namespace
```<br><br>**JSON**<br><br>```json
{
  "type": "Namespace"
}
``` |

api_version	
description	Top-level attribute that specifies the Sensu API group and version. The `api_version` should always be `core/v2`.
required	true
type	String

example	YML
| | ```
api_version: core/v2
``` |
| | **JSON** |
| | ```
{
  "api_version": "core/v2"
}
``` |

metadata

description	Top-level collection of metadata about the namespace. For namespaces, the metatdata should always be empty.
required	true
type	Map of key-value pairs
example	YML
| | ```
metadata: {}
``` |
| | **JSON** |
| | ```
{
  "metadata": {}
}
``` |

spec

description	Top-level map that includes the namespace's spec attributes.
required	true

type	Map of key-value pairs

YML

example	

```yaml
spec:
  name: production
```

JSON

```json
{
  "spec": {
    "name": "production"
  }
}
```

Spec 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

YML

example	

```yaml
name: production
```

JSON

```json
{
  "name": "production"
}
```

Role-based access control (RBAC) reference

Sensu's role-based access control (RBAC) helps different teams and projects share a Sensu instance. RBAC allows you to specify actions users are allowed to take against resources, within namespaces or across all namespaces, based on roles bound to the user or to one or more groups the user is a member of.

- **Roles** create sets of permissions (for example, get and delete) tied to resource types. **Cluster roles** apply permissions across namespaces and include access to cluster-wide resources like users and namespaces.
- **Users** represent a person or agent that interacts with Sensu. Users can belong to one or more **groups**.
- **Role bindings** assign a role to a set of users and groups within a namespace. **Cluster role bindings** assign a cluster role to a set of users and groups cluster-wide.

RBAC configuration applies to sensuctl, the API, and the web UI.

Resources

Permissions within Sensu 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`	Dynamic runtime asset resources within a namespace
`checks`	Check resources within a namespace

`entities`	[Entity](#) resources within a namespace
`events`	[Event](#) resources within a namespace
`extensions`	Placeholder type
`filters`	[Filter](#) resources within a namespace
`handlers`	[Handler](#) resources within a namespace
`hooks`	[Hook](#) resources within a namespace
`mutators`	[Mutator](#) resources within a namespace
`rolebindings`	Namespace-specific role assigners
`roles`	Namespace-specific permission sets
`searches`	Saved [web UI](#) search queries
`secrets`	[Secrets](#) (for example, username or password)
`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
`apikeys`	[Persistent universally unique identifier (UUID)](#) for authentication
`authproviders`	[Authentication provider](#) configuration
`clusterrolebindings`	Cluster-wide role assigners
`clusterroles`	Cluster-wide permission sets
`clusters`	Sensu clusters running multiple [Sensu backends](#)

`config`	Global configuration for web UI display
`etcd-replicators`	Mirror RBAC resource changes to follower clusters
`license`	Sensu commercial license
`namespaces`	Resource partitions within a Sensu instance
`provider`	PostgreSQL event store provider
`providers`	Secrets providers
`users`	People or agents that interact with Sensu

Special resource types

You can 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 configure sensuctl or log in to the web UI.

User example

The following example shows a user resource definition:

YML

```yaml
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  groups:
  - ops
  - dev
  password: USER_PASSWORD
  password_hash: $5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm
  username: alice
```

JSON

```json
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "username": "alice",
    "password": "USER_PASSWORD",
    "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm",
    "disabled": false,
    "groups": ["ops", "dev"]
  }
}
```

To create this user with `sensuctl create` , first save the definition to a file like `users.yml` or `users.json` .

Then, run:

SHELL

```shell
sensuctl create --file users.yml
```

SHELL

```
sensuctl create --file users.json
```

Default users

During the Sensu backend installation process, you create an administrator username and password and a `default` namespace.

This is the admin 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`

After you configure sensuctl, you can change the admin user's password with the `change-password` command.

Sensu also creates a default `agent` user with the password `P@ssw0rd!`. This user/password combination corresponds to the defaults the Sensu agent uses. You can configure the Sensu agent's user credentials with the `user` and `password` agent configuration flags.

Manage users

To test the password for a user created with Sensu's built-in basic authentication:

```
sensuctl user test-creds USERNAME --password 'password'
```

An empty response indicates valid credentials. A `request-unauthorized` response indicates invalid credentials.

> **NOTE**: The `sensuctl user test-creds` command tests passwords for users created with Sensu's built-in basic authentication provider. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).

To change the password for a user:

```
sensuctl user change-password USERNAME --current-password CURRENT_PASSWORD --new-password NEW_PASSWORD
```

You can also use sensuctl to reset a user's password or generate a password hash.

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 list all users within Sensu.

To return a list of users in `yaml` or `wrapped-json` format for use with `sensuctl create`:

SHELL

```
sensuctl user list --format yaml
```

SHELL

```
sensuctl user list --format wrapped-json
```

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
```

You can create any number of users, each with their own passwords.Users are granted permissions by role bindings or cluster role bindings, but as a general rule, users have no permissions by default.

By default, the agent user belongs to the `system:agent` group.The `system:agent` cluster role binding grants the `system:agent` cluster role to the members of this group.To grant agent users the permissions they need to report events into any namespace, add agent users to the `system:agent` group.

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

YML

example	

```yaml
username: alice
```

JSON

```json
{
  "username": "alice"
}
```

password

description	User's password. Passwords must have at least eight characters.

> **NOTE**: You only need to set either the `password` or the `password_hash` (not both). We recommend using the `password_hash` because it eliminates the need to store cleartext passwords.

required	true

type	String

YML

example	

```yaml
password: USER_PASSWORD
```

JSON

```json
{
  "password": "USER_PASSWORD"
}
```

groups

description	Groups to which the user belongs.
required	false
type	Array

example

YML

```yml
groups:
- dev
- ops
```

JSON

```json
{
  "groups": [
    "dev",
    "ops"
  ]
}
```

disabled

description	If `true`, the user's account is disabled. Otherwise, `false`.
required	false
type	Boolean
default	`false`

example

YML

```yml
disabled: false
```

JSON

```json
{
```

```
      "disabled": false
   }
```

password_hash

description	Bcrypt password hash. You can use the `password_hash` in your user definitions instead of storing cleartext passwords.
	NOTE: You only need to set either the `password` or the `password_hash` (not both). We recommend using the `password_hash` because it eliminates the need to store cleartext passwords.

required	false

type	String

YML

example	

```
password_hash:
$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyN
m
```

JSON

```
{
   "password_hash":
"$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAy
Nm"
}
```

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 roles 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 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`.

Role example

The following example shows a role resource definition:

YML

```
---
type: Role
api_version: core/v2
metadata:
```

```yaml
  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

```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"]
```

```
        }
      ]
    }
  }
```

To create this role with `sensuctl create`, first save the definition to a file like `roles.yml` or `roles.json`.

Then, run:

SHELL

```
sensuctl create --file roles.yml
```

SHELL

```
sensuctl create --file roles.json
```

Cluster role example

The following example shows a cluster role resource definition:

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  name: all-resources-all-verbs
spec:
  rules:
  - 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"]
      }
    ]
  }
}
```

```
    }
```

To create this cluster role with `sensuctl create` , first save the definition to a file like `cluster_roles.yml` or `cluster_roles.json` .

Then, run:

SHELL

```
sensuctl create --file cluster_roles.yml
```

SHELL

```
sensuctl create --file cluster_roles.json
```

Default roles and cluster roles

Every Sensu backend includes:

role name	type	description
`system:pipeline`	`Role`	Facility that allows the EventFilter engine to load events from Sensu's event store. `system:pipeline` is an implementation detail and should not be assigned to Sensu users.
`cluster-admin`	`Cluster Role`	Full access to all resource types across namespaces, including access to cluster-wide resource types.
`admin`	`Cluster Role`	Full access to all resource types. You can apply this cluster role within a namespace by using a role binding (not a cluster role binding).
`edit`	`Cluster Rol`	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).

	e	

`view`	`Clu`
	`ster`
	`Rol`
	`e`
		Read-only permission to most <u>resource types</u> 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`	`Clu`
	`ster`
	`Rol`
	`e`
		Used internally by Sensu agents. You can configure an agent's user credentials using the `user` and `password` <u>agent configuration flags</u>.

`system:user`	`Clu`
	`ster`
	`Rol`
	`e`
		Get and update permissions for local resources for the current user.

Manage roles and cluster roles

You can use <u>sensuctl</u> 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 <u>sensuctl</u> to list all roles within Sensu:

```
sensuctl role list
```

To review 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
```

This command creates the following role resource definition:

YML

```yml
---
type: Role
api_version: core/v2
metadata:
  created_by: admin
  name: prod-admin
  namespace: production
spec:
  rules:
  - resource_names: null
    resources:
    - '*'
```

```
      verbs:
      - get
      - list
      - create
      - update
      - delete
```

JSON

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "prod-admin",
    "namespace": "production"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "*"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

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
```

This command creates the following role binding resource definition:

YML

```yaml
---
type: RoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: prod-admin-oncall
  namespace: production
spec:
  role_ref:
    name: prod-admin
    type: Role
  subjects:
  - name: oncall
    type: Group
```

JSON

```json
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "prod-admin-oncall",
    "namespace": "production"
  },
  "spec": {
    "role_ref": {
      "name": "prod-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "oncall",
        "type": "Group"
```

```
      }
    ]
  }
}
```

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'
```

This command creates the following cluster-wide role resource definition:

YML

```yaml
---
type: ClusterRole
api_version: core/v2
metadata:
  created_by: admin
  name: global-event-reader
spec:
  rules:
  - resource_names: null
    resources:
    - events
    verbs:
    - get
    - list
```

JSON

```json
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
```

```
      "created_by": "admin",
      "name": "global-event-reader"
    },
    "spec": {
      "rules": [
        {
          "resource_names": null,
          "resources": [
            "events"
          ],
          "verbs": [
            "get",
            "list"
          ]
        }
      ]
    }
  }
```

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
	YML

example	
	`name: admin`

JSON

```
{
  "name": "admin"
}
```

namespace

description	Namespace the role is restricted to. This attribute is not available for cluster roles.
required	false
type	String

YML

```
namespace: production
```

example

JSON

```
{
  "namespace": "production"
}
```

rules

description	Rulesets that the role applies.
required	true
type	Array

example	**YML**
| | ```yaml
rules:
- verbs:
  - get
  - list
  resources:
  - checks
  resource_names:
  - ''
``` |
| | **JSON** |
| | ```json
{
  "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	YML

```yaml
verbs:
- get
- list
```

JSON

```json
{
  "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. Read resource types to learn about available types.
required	true
type	Array

example	YML

```yaml
resources:
- checks
```

JSON

```json
{
```

```
    "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	**YML**

```yaml
resource_names:
- check-cpu
```

JSON

```json
{
  "resource_names": [
    "check-cpu"
  ]
}
``` |

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` .

Make sure to include the groups prefix and username prefix for the authentication provider when creating Sensu role bindings and cluster role bindings.Without an assigned role or cluster role, users can sign in to the web UI but can't access any Sensu resources.With the correct roles and bindings configured, users can log in to sensuctl and the web UI using their single-sign-on username and password (no prefixes required).

Role binding example

The following example shows a role binding resource definition:

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": [
        {
          "name": "bob",
          "type": "User"
        }
      ]
    }
  }
```

To create this role binding with sensuctl create , first save the definition to a file like
 rolebindings.yml or rolebindings.json .

Then, run:

SHELL

```
sensuctl create --file rolebindings.yml
```

SHELL

```
sensuctl create --file rolebindings.json
```

Cluster role binding example

The following example shows a cluster role binding resource definition:

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: cluster-admin
```

```yaml
  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"
      }
    ]
  }
}
```

To create this cluster role binding with `sensuctl create`, first save the definition to a file like `clusterrolebindings.yml` or `clusterrolebindings.json`.

Then, run:

SHELL

```shell
sensuctl create --file clusterrolebindings.yml
```

SHELL

```
sensuctl create --file clusterrolebindings.json
```

Default role bindings and cluster role bindings

Every Sensu backend includes:

role name	type	description
`system:pipeline`	`Role Binding`	Facility that allows the EventFilter engine to load events from Sensu's event store. `system:pipeline` is an implementation detail and should not be applied to Sensu users.
`cluster-admin`	`ClusterRoleBinding`	Full access to all resource types across namespaces, including access to cluster-wide resource types.
`system:agent`	`ClusterRoleBinding`	Full access to all events. Used internally by Sensu agents.
`system:user`	`ClusterRoleBinding`	Get and update permissions for local resources for the current user.

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 list all role bindings within Sensu:

```
sensuctl role-binding list
```

To review the details for a specific role binding:

```
sensuctl role-binding info [BINDING-NAME]
```

To list cluster role bindings:

```
sensuctl cluster-role-binding list
```

Create role bindings and cluster role bindings

You can use sensuctl to create a role binding that assigns a role:

```
sensuctl role-binding create NAME --role=NAME --user=username --group=groupname
```

This command creates the following role binding resource definition:

YML

```
---
type: RoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: NAME
```

```
    namespace: default
  spec:
    role_ref:
      name: NAME
      type: Role
    subjects:
    - name: groupname
      type: Group
    - name: username
      type: User
```

JSON

```
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "NAME",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "NAME",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "groupname",
        "type": "Group"
      },
      {
        "name": "username",
        "type": "User"
      }
    ]
  }
}
```

To create a role binding that assigns a cluster role:

```
sensuctl role-binding create NAME --cluster-role=NAME --user=username --
group=groupname
```

This command creates the following role binding resource definition:

YML

```yaml
---
type: RoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: NAME
  namespace: default
spec:
  role_ref:
    name: NAME
    type: ClusterRole
  subjects:
  - name: groupname
    type: Group
  - name: username
    type: User
```

JSON

```json
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "NAME",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "NAME",
      "type": "ClusterRole"
    },
```

```json
      "subjects": [
        {
          "name": "groupname",
          "type": "Group"
        },
        {
          "name": "username",
          "type": "User"
        }
      ]
    }
  }
```

To create a cluster role binding:

```
sensuctl cluster-role-binding create NAME --cluster-role=NAME --user=username --
group=groupname
```

This command creates the following cluster role binding resource definition:

YML

```yaml
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  created_by: admin
  name: NAME
spec:
  role_ref:
    name: NAME
    type: ClusterRole
  subjects:
  - name: groupname
    type: Group
  - name: username
    type: User
```

JSON

```json
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "NAME"
  },
  "spec": {
    "role_ref": {
      "name": "NAME",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "groupname",
        "type": "Group"
      },
      {
        "name": "username",
        "type": "User"
      }
    ]
  }
}
```

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

YML

```yaml
roleRef:
  type: Role
  name: event-reader
```

JSON

```json
{
  "roleRef": {
    "type": "Role",
    "name": "event-reader"
  }
}
```

example	(see above)

subjects

description	Users or groups being assigned.
required	true
type	Array

YML

```yaml
subjects:
- type: User
  name: alice
```

JSON

```json
{
  "subjects": [
    {
```

example	(see above)

```
        "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	**YML**

```
  type: Role
```

JSON

```
  {
    "type": "Role"
  }
```

name	
description	Name of the role or cluster role being assigned.
required	true
type	String
example	**YML**

```
    name: event-reader
```

JSON

```json
{
  "name": "event-reader"
}
```

`subjects` *specification*

type	
description	`User` for assigning a user or `Group` for assigning a group.
required	true
type	String

YML

```
    type: User
```

JSON

```json
{
  "type": "User"
}
```

name	
description	Username or group name.
required	true
type	String

example	**YML** **name:** alice

	JSON
	
	{
	  "name": "alice"
	}

example with prefix	**YML** **name:** ad:alice

	JSON
	
	{
	  "name": "ad:alice"
	}

Create a role and role binding

The following role and role binding give a `dev` group access to create and manage Sensu workflows within the `default` namespace.

YML

```yml
---
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
```

JSON

```
{
  "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"]
      }
    ]
  }
}
```

YML

```
---
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
```

JSON

```json
{
  "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"
      }
    ]
  }
}
```

Create a role and role binding with a group prefix

In this example, if a groups_prefix of `ad` is configured for Active Directory authentication, the role and role binding will give a `dev` group access to create and manage Sensu workflows within the

`default` namespace.

YML

```yaml
---
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
```

JSON

```json
{
  "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"]
        }
    ]
}
}
```

YML

```yaml
---
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
```

JSON

```json
{
  "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"
```

```
        }
    ]
  }
}
```

Assign user permissions within a namespace

To assign permissions to a user:

1. Create the user.
2. Create a role.
3. Create a role binding to assign the role to the user.

For example, the following configuration creates a user `alice`, a role `default-admin`, and a role binding `alice-default-admin`, giving `alice` full permissions for namespaced resource types within the `default` namespace. You can add these resources to Sensu using `sensuctl create`.

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice"
  }
}
```

YML

```yaml
---
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
    - searches
    - silenced
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```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", "searches", "silenced"
          ],
          "verbs": ["get", "list", "create", "update", "delete"]
        }
      ]
    }
  }
```

YML

```
---
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
```

JSON

```
{
  "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`.

YML

```
---
type: User
api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
```

JSON

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

```
    "metadata": {},
    "spec": {
      "disabled": false,
      "username": "alice"
    }
  }
```

YML

```yaml
---
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
    - searches
    - silenced
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
```

```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", "searches", "silenced"
          ],
          "verbs": ["get", "list", "create", "update", "delete"]
        }
      ]
    }
  }
```

YML

```yaml
---
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
```

JSON

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

```json
    "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`.

YML

```yaml
---
type: User
```

```
  api_version: core/v2
metadata: {}
spec:
  disabled: false
  username: alice
  groups:
  - ops
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice",
    "groups": ["ops"]
  }
}
```

YML

```
---
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
```

JSON

```
{
  "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"]
      }
    ]
  }
}
```

YML

```
---
type: ClusterRoleBinding
api_version: core/v2
metadata:
  name: ops-default-admin
spec:
  role_ref:
    name: default-admin
    type: ClusterRole
  subjects:
  - name: ops
    type: Group
```

JSON

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-default-admin"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

Reuse cluster roles across namespaces

Reusing cluster roles across namespaces can reduce the number of resources you need to manage.

For example, suppose you have three teams, each with its own namespace. You write a script that uses limited service accounts to create and delete silences. You want to use the script for all three team namespaces, so you create a role with the required permissions and a role binding in each namespace: six new resources. If you need to change the permissions for the script, you will need to update each role in the team namespaces (three resources).

A better approach is to create a single cluster role that grants the required permissions, plus one role binding in each namespace to tie the permissions to the namespace's limited service account. With this configuration, you only need to update one resource to make permission changes: the `silencing-script` cluster role. Sensu will automatically apply updates in each team's namespace using the role bindings that define each limited service account as a subject of the cluster role.

1. Create a limited service account user in each namespace:

```
sensuctl user create silencing-service-team-1 --password='password'
```

This creates the following user definition:

YML

```
---
type: User
api_version: core/v2
metadata:
  name: silencing-service-team-1
spec:
  disabled: false
  username: silencing-service-team-1
```

JSON

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {
    "name": "silencing-service-team-1"
  },
  "spec": {
    "disabled": false,
```

```
      "username": "silencing-service-team-1"
    }
  }
```

Repeat this step to create a limited service account user in each team's namespace.

2. Create a <u>cluster role</u> with get, list, create, update, and delete permissions for silences:

```
sensuctl cluster-role create silencing-script --verb
get,list,create,update,delete --resource silenced
```

This command creates the cluster role that has the permissions the silencing service accounts will need:

YML

```
---
type: ClusterRole
api_version: core/v2
metadata:
  created_by: admin
  name: silencing-script
spec:
  rules:
  - resource_names: null
    resources:
    - silenced
    verbs:
    - get
    - list
    - create
    - update
    - delete
```

JSON

```
{
  "type": "ClusterRole",
```

```
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "silencing-script"
  },
  "spec": {
    "rules": [
      {
        "resource_names": null,
        "resources": [
          "silenced"
        ],
        "verbs": [
          "get",
          "list",
          "create",
          "update",
          "delete"
        ]
      }
    ]
  }
}
```

3. Create a role binding in each team namespace to assign the `silencing-script` cluster role to the team's `silencing-service` user. For example, use this command to create the role binding for Team 1:

```
sensuctl role-binding create silencing-script-binding-team-1 --cluster-role
silencing-script --user silencing-service-team-1 --namespace team1
```

This command creates the role binding that ties the correct permissions (via the `silencing-script` cluster role) with your service account (via the user `silencing-service-team-1`):

YML

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

```yaml
metadata:
  created_by: admin
  name: silencing-script-binding-team-1
  namespace: team1
spec:
  role_ref:
    name: silencing-script
    type: ClusterRole
  subjects:
  - name: silencing-service-team-1
    type: User
```

JSON

```json
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "silencing-script-binding-team-1",
    "namespace": "team1"
  },
  "spec": {
    "role_ref": {
      "name": "silencing-script",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "silencing-service-team-1",
        "type": "User"
      }
    ]
  }
}
```

Repeat this step to create a role binding for the `silencing-script` cluster role and the limited service account user in each team's namespace.

Maintain Sensu

The Maintain Sensu category includes information to keep your Sensu installation up-to-date and running smoothly.

Upgrade or migrate

Follow the upgrade guide for step-by-step instructions to upgrade to the latest version of Sensu from any earlier version. The upgrade instructions include details about important changes between versions that could affect your upgrade and any special requirements to make sure your upgrade is successful.

If you are still using Sensu Core or Sensu Enterprise, follow Migrate from Sensu Core and Sensu Enterprise to Sensu Go to upgrade to Sensu Go. The migrate guide includes links to Sensu's migration resources and Core and Enterprise configuration translation tools, as well as instructions for installing Sensu Go alongside your existing Sensu Core or Enterprise instance.

Troubleshoot

Use the Sensu troubleshooting guide to diagnose and resolve common problems, and read about tuning options for specific performance issues. Learn how to read, configure, and find the logs produced by Sensu services. Sensu log messages can help you identify and solve backend startup errors and permissions issues.

The troubleshooting guide also describes how to use Sensu handlers and filters to test and debug your observability pipeline and diagnose problems related to dynamic runtime assets.

Manage license

Read the license reference to learn how to activate your commercial license. The license reference also explains how to view your license details and expiration date and find your current entity count and limits.

Upgrade Sensu

To upgrade to the latest version of Sensu Go:

1. <u>Install</u> the latest packages or Docker image.

2. For systems that use `systemd`, run:

   ```
   sudo systemctl daemon-reload
   ```

3. Restart the Sensu agent:

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

4. Restart the Sensu backend:

   ```
   sudo service sensu-backend restart
   ```

5. Run a single upgrade command on one your Sensu backends to migrate the cluster:

   ```
   sensu-backend upgrade
   ```

   To skip confirmation and immediately run the upgrade command, add the `--skip-confirm` flag:

   ```
   sensu-backend upgrade --skip-confirm
   ```

   NOTE: *If you are deploying a new Sensu cluster rather than upgrading from a previous*

> version, you do not need to run the `sensu-backend upgrade` command.

6. Enter `y` or `n` to confirm if you did *not* add the `--skip-confirm` flag in step 5. Otherwise, skip this step.

7. Wait a few seconds for the upgrade command to run. You may notice some inconsistencies in your entity list until the cluster finishes upgrading. Despite this, your cluster will continue to publish standard check requests and process events.

   If you run the upgrade command more than once, it will not harm the cluster — you'll just receive a response that the upgrade command has already been run.

   Some minor versions do not involve database-specific changes, and the `sensu-backend upgrade` tool will report that nothing was upgraded. Check the release notes to confirm whether a version has database-specific changes that require a backend upgrade.

8. Confirm the installed version for the agent, backend, and sensuctl:

```
sensu-agent version
```

```
sensu-backend version
```

```
sensuctl version
```

> **PRO TIP**: If your upgrade is unsuccessful, read the version-specific information on this page and complete the instructions for each version, starting with your current version and continuing up to the version you want to install.
>
> For example, to debug an upgrade from 5.5.0 to 6.1.0, start with Upgrade Sensu clusters from 5.7.0 or earlier to 5.8.0 or later.

Upgrade to Sensu Go 6.1.0 from 6.0.0

If you are using 6.0.0 and have a large number of events in PostgreSQL, you may experience a short

period of unavailability after you upgrade to 6.1.0. This pause will occur while the optimized selector information is populating during automatic database migration. It may last for a period of a few seconds to a few minutes.

This pause may extend to API request processing, so sensuctl and the web UI may also be unavailable during the migration.

Upgrade to Sensu Go 6.0 from a 5.x deployment

Before you upgrade to Sensu 6.0, use `sensuctl dump` to create a backup of your existing installation.

You will not be able to downgrade to a Sensu 5.x version after you upgrade your database to Sensu 6.0 after you restart the backend in the upgrade process.

Upgrade to Sensu Go 5.16.0 from any earlier version

As of Sensu Go 5.16.0, Sensu's free entity limit is 100 entities. All commercial features are available for free in the packaged Sensu Go distribution for up to 100 entities.

When you upgrade to 5.16.0, if your existing unlicensed instance has more than 100 entities, Sensu will continue to monitor those entities. However, if you try to create any new entities via the HTTP API or sensuctl, you will receive the following message:

 This functionality requires a valid Sensu Go license with a sufficient entity limit. To get a valid license file, arrange a trial, or increase your entity limit, contact Sales.

Connections from new agents will fail and result in a log message like this:

```
{"component":"agent","error":"handshake failed with status
402","level":"error","msg":"reconnection attempt failed","time":"2019-11-
20T05:49:24-07:00"}
```

In the web UI, you will receive the following message when you reach the 100-entity limit:

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

Upgrade Sensu clusters from 5.7.0 or earlier to 5.8.0 or later

> *NOTE: This section applies only to Sensu clusters with multiple backend nodes.*

Due to updates to etcd serialization, you must shut down Sensu clusters with multiple backend nodes while upgrading from Sensu Go 5.7.0 or earlier to 5.8.0 or later.Read the backend reference for more information about stopping and starting backends.

Upgrade Sensu backend binaries to 5.1.0

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

For Sensu backend binaries, the default `state-dir` in 5.1.0 is now `/var/lib/sensu/sensu-backend` instead of `/var/lib/sensu`.To upgrade your Sensu backend binary to 5.1.0, first download the latest version.Then, make sure the `/etc/sensu/backend.yml` configuration file specifies a `state-dir`.To continue using `/var/lib/sensu` as the `state-dir` to store backend data, add the following configuration to `/etc/sensu/backend.yml`:

```
state-dir: "/var/lib/sensu"
```

Then restart the backend:

```
sudo service sensu-backend restart
```

Migrate from Sensu Core and Sensu Enterprise to Sensu Go

This guide includes general information for migrating your Sensu instance from Sensu Core and Sensu Enterprise to Sensu Go. For instructions and tools to help you translate your Sensu configuration from Sensu Core and Enterprise to Sensu Go, review the Sensu Translator project and our blog post about check configuration upgrades with the Sensu Go sandbox.

> **NOTE**: The information in this guide applies to Sensu Enterprise as well as Sensu Core, although we refer to "Sensu Core" for brevity. One step applies to Sensu Enterprise (and not Sensu Core) — it is designated as Sensu Enterprise-only.

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

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

> **WARNING**: To install Sensu Go alongside your current Sensu instance, you must upgrade to at least Sensu Core 1.9.0-2. If you need to upgrade, contact Sensu.

Aside from this migration guide, these resources can help you migrate from Sensu Core or Sensu Enterprise to Sensu Go:

- **Sensu Community Slack**: Join hundreds of other Sensu users in our Community Slack, where you can ask questions and benefit from tips others picked up during their own Sensu Go migrations.

- **Sensu Community Forum**: Drop a question in our dedicated category for migrating to Go.

- **Sensu Go Sandbox**: Download the sandbox and try out some monitoring workflows with Sensu Go.

- **Sensu Translator**: Use this command-line tool to generate Sensu Go configurations from your Sensu Core config files.

We also offer **commercial support** and **professional services** packages to help with your Sensu Go migration.

Configuration management with Ansible, Chef, and Puppet

Configuration management integrations for Sensu Go are available for Ansible, Chef, and Puppet:

- Ansible collection for Sensu Go and documentation site
- Chef cookbook for Sensu Go — contact us for more information
- Puppet module for Sensu Go

Packaging

Sensu Go is provided as three packages: sensu-go-backend, sensu-go-agent, and sensu-go-cli (sensuctl).This is a fundamental change in Sensu terminology from Sensu Core: the server is now the backend.

Clients are represented within Sensu Go as abstract entities that can describe a wider range of system components such as network gear, a web server, or a cloud resource.Entities include agent entities that run a Sensu agent and the familiar proxy entities.

Read Sensu concepts and terminology to learn more about new terms in Sensu Go.

Architecture

The external RabbitMQ transport and Redis datastore in Sensu Core are replaced with an embedded transport and etcd datastore in Sensu Go.

WORKSTATION		SENSU-BACKEND		MONITORED ENTITIES
sensuctl	→	HTTP API	WebSocket API	requests checks → sensu-agent
		PORT 8080	PORT 8081	← returns events
browser	→	web UI	embedded etcd store	
		PORT 3000		
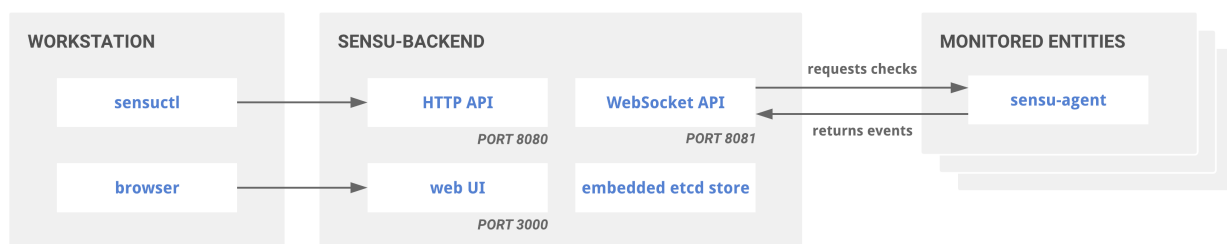
In Sensu Go, the Sensu backend and agent are configured with YAML files or the `sensu-backend` or `sensu-agent` command line tools rather than JSON files.Sensu checks and pipeline elements are configured via the API or sensuctl tool in Sensu Go instead of JSON files.

The **Sensu backend** is powered by an embedded transport and <u>etcd</u> datastore and gives you flexible, automated workflows to route metrics and alerts.Sensu backends require persistent storage for their embedded database, disk space for local dynamic runtime asset caching, and several exposed ports:

- ¬ 2379 (gRPC) Sensu storage client: Required for Sensu backends using an external etcd instance

- ¬ 2380 (gRPC) Sensu storage peer: Required for other Sensu backends in a <u>cluster</u>

- ¬ 3000 (HTTP/HTTPS) <u>Sensu web UI</u>: Required for all Sensu backends using a Sensu web UI

- ¬ 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

Sensu agents are lightweight clients that run on the infrastructure components you want to monitor. Agents automatically register with Sensu as entities and are responsible for creating check and metric events to send to the backend event pipeline.

The Sensu agent uses:

- ¬ 3030 (TCP/UDP) Sensu agent socket: Required for Sensu agents using the agent socket

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

- ¬ 8125 (UDP) <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>.

Agents that use Sensu <u>dynamic runtime assets</u> require some disk space for a local cache.

Read the <u>backend</u>, <u>agent</u>, and <u>sensuctl</u> 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**. read 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), dynamic runtime 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 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

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.

Ruby eval logic from Sensu Core 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, **Sensu Go does not include the built-in occurrence-based event filter in Sensu Core**, which allowed you to control the number of duplicate events that reached the handler. You can replicate the occurrence-based filter's functionality with Sensu Go's [repeated events filter definition](#).

Fatigue check filter

For Sensu Go users, we recommend the fatigue check filter, a JavaScript implementation of the `occurrences` filter from Sensu Core. This filter looks for check and entity annotations in each event it receives and uses the values of those annotations to configure the filter's behavior on a per-event basis.

The Sensu Translator version 1.1.0 retrieves occurrence and refresh values from a Sensu Core check definition and outputs them as annotations in a Sensu Go check definition, compatible with the fatigue check filter.

However, the Sensu Translator doesn't automatically add the fatigue check filter dynamic runtime asset or the filter configuration you need to run it. To use the fatigue check filter dynamic runtime asset, you must register it, create a correctly configured event filter definition, and add the event filter to the list of filters on applicable handlers.

Dynamic runtime assets

The `sensu-install` tool in Sensu Core is replaced by assets in Sensu Go. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

You can still install Sensu Community plugins in Ruby via `sensu-install` by installing sensu-plugins-ruby. Read Install plugins for more information.

Role-based access control (RBAC)

Role-based access control (RBAC) is a built-in feature of the open-source version of Sensu Go. RBAC allows you to manage and access users and resources based on namespaces, groups, roles, and bindings. To set up RBAC in Sensu Go, read the RBAC reference and Create a read-only user.

Silencing

Silencing is disabled by default in Sensu Go. You must explicitly enable silencing 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).

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. Read the API overview for more information.

Step-by-step migration instructions

Step 1: Install Sensu Go

1. Install the Sensu Go backend

The Sensu backend is available for Ubuntu/Debian, RHEL/CentOS, and Docker. Read the installation guide to install, configure, and start the Sensu backend according to your deployment strategy.

2. Log in to the Sensu web UI

The Sensu Go web UI provides a unified view of your observability events with user-friendly tools to reduce alert fatigue and manage your Sensu instance. After starting the Sensu backend, open the web UI by visiting http://localhost:3000. You may need to replace `localhost` with the hostname or IP address where the Sensu backend is running.

To log in, enter your Sensu user credentials, or use Sensu's default admin credentials (username: `admin` and password: `P@ssw0rd!`).

3. Install sensuctl on your workstation

Sensuctl is a command line tool for managing resources within Sensu. It works by calling Sensu's HTTP API to create, read, update, and delete resources, events, and entities. Sensuctl is available for Linux, Windows, and macOS. Read the installation guide to install and configure sensuctl.

4. Set up Sensu users

Role-based access control (RBAC) is a built-in feature of the open-source version of Sensu Go. RBAC allows you to manage and access users and resources based on namespaces, groups, roles, and bindings. To set up RBAC in Sensu Go, read the RBAC reference and Create a read-only user.

In Sensu Go, namespaces partition resources within a Sensu instance. Sensu Go entities, checks, handlers, and other namespaced resources belong to a single namespace. The Sensu translator places all translated resources into the `default` namespace — we'll use the translater in a moment.

In addition to built-in RBAC, Sensu Go's commercial features include support for authentication using Microsoft Active Directory (AD) and standards-compliant Lightweight Directory Access Protocol tools like OpenLDAP.

5. Install agents

The Sensu agent is available for Ubuntu/Debian, RHEL/CentOS, Windows, and Docker. Read the installation guide to install, configure, and start Sensu agents.

If you're doing a side-by-side migration, add `api-port` (default: `3031`) and `socket-port` (default: `3030`) to your agent configuration (`/etc/sensu/agent.yml`). This prevents the Sensu Go agent API and socket from conflicting with the Sensu Core client API and socket.

```
api-port: 3031
socket-port: 3030
```

You can also disable these features in the agent configuration using the `disable-socket` and `disable-api` flags.

Sensu should now be installed and functional. The next step is to translate your Sensu Core configuration to Sensu Go.

Step 2: Translate your configuration

Use the Sensu Translator command line tool to transfer your Sensu Core checks, handlers, and mutators to Sensu Go.

1. Run the translator

Install dependencies:

```
yum install -q -y rubygems ruby-devel
```

Install the Sensu translator:

```
gem install sensu-translator
```

Run the Sensu translator to translate all configuration in /etc/sensu/conf.d to Sensu Go and output to /sensu_config_translated:

```
sensu-translator -d /etc/sensu/conf.d -o /sensu_config_translated
```

As an option, you can also translate your configuration in sections according to resource type.

If translation is successful, you should receive a few callouts followed by `DONE!`, similar to this example:

```
Sensu 1.x filter translation is not yet supported
Unable to translate Sensu 1.x filter: only_production
{:attributes=>{:check=>{:environment=>"production"}}}
DONE!
```

Combine your config into a sensuctl-readable format.

NOTE: for use with `sensuctl create`, *do not use a comma between resource objects in Sensu Go resource definitions in JSON format.*

```
find sensu_config_translated/ -name '*.json' -exec cat {} \; >
sensu_config_translated_singlefile.json
```

Most attributes are ready to use as-is, but you'll need to adjust your Sensu Go configuration manually to migrate some of Sensu's features.

> **NOTE**: To streamline a comparison of your Sensu Core configuration with your Sensu Go configuration, output your current Sensu Core configuration using the API: `curl -s http://127.0.0.1:4567/settings | jq . > sensu_config_original.json`.

2. Translate checks

Review your Sensu Core check configuration for the following attributes, and make the corresponding updates to your Sensu Go configuration.

Core attribute	Manual updates required in Sensu Go config
`::: foo :::`	Update the syntax for token substitution from triple colons to double curly braces. For example: `{{ foo }}`
`stdin: true`	No updates required. Sensu Go checks accept data on stdin by default.
`handlers: default`	Sensu Go does not have a default handler. Create a handler named `default` to continue using this pattern.
`subdues`	Check subdues are not available in Sensu Go.
`standalone: true`	Standalone checks are not supported in Sensu Go, although you can achieve similar functionality using role-based access control, dynamic runtime assets, and entity subscriptions. The translator assigns all Core standalone checks to a `standalone` subscription in Sensu Go. Configure one or more Sensu Go agents with the `standalone` subscription to execute formerly standalone checks.
`metrics: true`	Review the translate metric checks section.
`proxy_requests`	Review the translate proxy requests section.

`subscribers:` `roundrobin...`	Remove `roundrobin` from the subscription name, and add the `round_robin` check attribute set to `true`.
`aggregate`	Check aggregates are supported through the commercial Sensu Go Aggregate Check Plugin.
`hooks`	Review the translate hooks section.
`dependencies`	Use the Core Dependencies Filter dynamic runtime asset.

> **PRO TIP**: *When using token substitution in Sensu Go and accessing labels or annotations that include* `.` *(for example:* `sensu.io.json_attributes`*), use the* `index` *function. For example,* `{{index .annotations "web_url"}}` *substitutes the value of the* `web_url` *annotation;* `{{index .annotations "production.ID"}}` *substitutes the value of the* `production.ID` *annotation.*

Translate metric checks

The Sensu Core `type: metric` attribute is not part of the Sensu Go check spec, so you'll need to adjust it manually. Sensu Core checks could be configured as `type: metric`, which told Sensu to always handle the check regardless of the check status output. This allowed Sensu Core to process output metrics via a handler even when the check status was not in an alerting state.

Sensu Go treats output metrics as first-class objects, so you can process check status as well as output metrics via different event pipelines. Read the guide to metric output to update your metric checks with the `output_metric_handlers` and `output_metric_format` attributes and use `output_metric_tags` to enrich extracted metrics output.

Translate proxy requests and proxy entities

Read Monitor external resources to re-configure `proxy_requests` attributes and update your proxy check configuration. Read the entity reference to re-create your proxy client configurations as Sensu Go proxy entities.

Translate hooks

Check hooks are now a resource type in Sensu Go, so you can create, manage, and reuse hooks independently of check definitions. You can also execute multiple hooks for any given response code. Read the guide and hooks reference docs to re-create your Sensu Core hooks as Sensu Go hook resources.

Custom attributes

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

The Sensu Translator stores all check extended attributes in the check metadata annotation named `sensu.io.json_attributes`. Read the checks reference for more information about using labels and annotations in check definitions.

3. Translate event filters

Ruby eval logic used in Sensu Core filters is replaced with JavaScript expressions in Sensu Go, opening up powerful possibilities to combine filters with filter dynamic runtime assets. As a result, you'll need to rewrite your Sensu Core filters in Sensu Go format.

First, review your Core handlers to identify which filters are being used. Then, follow the filter reference and guide to using filters to re-write your filters using Sensu Go expressions and event data. Check out the blog post on filters for a deep dive into Sensu Go filter capabilities.

Sensu Core hourly filter:

```
{
  "filters": {
    "recurrences": {
      "attributes": {
        "occurrences": "eval: value == 1 || value % 60 == 0"
      }
    }
  }
}
```

Sensu Go hourly filter:

YML

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

```
    created_by: admin
    name: hourly
    namespace: default
  spec:
    action: allow
    expressions:
    - event.check.occurrences == 1 || event.check.occurrences % (3600 /
  event.check.interval) == 0
    runtime_assets: null
```

JSON

```json
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "created_by": "admin",
    "name": "hourly",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1 || event.check.occurrences % (3600 /
  event.check.interval) == 0"
    ],
    "runtime_assets": null
  }
}
```

4. Translate handlers

In Sensu Go, all check results are considered events and are processed by event handlers.Use the built-in `is_incident` filter to recreate the Sensu Core behavior, in which only check results with a non-zero status are considered events.

> **NOTE**: Silencing is disabled by default in Sensu Go and must be explicitly enabled using the built-in `not_silenced` filter.Add the `not_silenced` filter to any handlers for which you want to enable Sensu's silencing feature.

Review your Sensu Core check configuration for the following attributes, and make the corresponding updates to your Sensu Go configuration.

Core attribute	Manual updates required in Sensu Go config
`filters: occurrences`	The built-in occurrences filter in Sensu Core is not available in Sensu Go, but you can replicate its functionality with the sensu-go-fatigue-check-filter asset.
`type: transport`	Transport handlers are not supported in 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: check_dependencies`	Sensu Go does not include a built-in check dependencies filter.
`severities`	Severities are not available in Sensu Go.
`handle_silenced`	Silencing is disabled by default in Sensu Go and must be explicitly enabled using the built-in `not_silenced` filter.
`handle_flapping`	All check results are considered events in Sensu Go and are processed by event handlers.

5. Upload your config to your Sensu Go instance

After you review your translated configuration, make any necessary updates, and add resource definitions for any filters and entities you want to migrate, you can upload your Sensu Go config using sensuctl.

```
sensuctl create --file /path/to/config.json
```

> **PRO TIP**: `sensuctl create` (and `sensuctl delete`) are powerful tools to help you manage your Sensu configs across namespaces. Read the [sensuctl reference][5] for more information.

Access your Sensu Go config using the Sensu API.

Set up a local API testing environment by saving your Sensu credentials and token as environment variables.This command requires curl and jq.

```
export SENSU_USER=admin && SENSU_PASS=P@ssw0rd!
export SENSU_TOKEN=`curl -XGET -u "$SENSU_USER:$SENSU_PASS" -s
http://localhost:8080/auth | jq -r ".access_token"`
```

Return a list of all configured checks:

```
curl -H "Authorization: Bearer $SENSU_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

Return a list of all configured handlers:

```
curl -H "Authorization: Bearer $SENSU_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers
```

You can also access your Sensu Go configuration in JSON or YAML using sensuctl.For example, `sensuctl check list --format wrapped-json` .Run `sensuctl help` To view available commands For more information about sensuctl's output formats (`json` , `wrapped-json` , and `yaml`), read the sensuctl reference.

Step 3: Translate plugins and register dynamic runtime assets

Sensu plugins

Within the Sensu Plugins org, review individual plugin READMEs for compatibility status with Sensu Go.For handler and mutators plugins, review the Sensu plugins README to map event data to the Sensu Go event format.This allows you to use Sensu plugins for handlers and mutators with Sensu Go without re-writing them.

To re-install Sensu plugins onto your Sensu Go agent nodes (check plugins) and backend nodes (mutator and handler plugins), read the guide to installing the `sensu-install` tool for use with Sensu Go.

Sensu Go dynamic runtime assets

The `sensu-install` tool in Sensu Core is replaced by <u>dynamic runtime assets</u> in Sensu Go.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Although dynamic runtime assets are not required to run Sensu Go, we recommend <u>using assets to install plugins</u> where possible.You can still install <u>Sensu Community plugins</u> in Ruby via `sensu-install` by installing <u>sensu-plugins-ruby</u>.Read <u>Install plugins</u> for more information.

Sensu supports dynamic runtime assets for checks, filters, mutators, and handlers.Discover, download, and share dynamic runtime assets with <u>Bonsai</u>, the Sensu asset hub.

To create your own dynamic runtime assets, read the <u>asset reference</u> and <u>guide to sharing an asset on Bonsai</u>.To contribute to converting a Sensu plugin to a dynamic runtime asset, read <u>the Discourse post</u>

Step 4: Translate Sensu Enterprise-only features

Integrations

Most Sensu Enterprise integrations as available as Sensu Go assets.Read the <u>guide to installing plugins with assets</u> to register assets with Sensu and update your Sensu Go handler definitions.

- <u>Chef</u>
- <u>Email</u>
- <u>Graphite</u>
- <u>InfluxDB</u>
- <u>IRC</u>
- <u>Jira</u>
- <u>PagerDuty</u>
- <u>ServiceNow</u>
- <u>Slack</u>
- <u>VictorOps</u>

Contact routing

Contact routing is available in Sensu Go using the has-contact filter asset.Read Route alerts with event filters to set up contact routing in Sensu Go.

LDAP

In addition to built-in RBAC, Sensu includes license-activated support for authentication using Microsoft Active Directory and standards-compliant Lightweight Directory Access Protocol tools like OpenLDAP.

Step 5: Sunset your Sensu Core instance

When you're ready to sunset your Sensu Core instance, stop the Sensu Core services according to the instructions for your platform — these instructions are listed under **Operating Sensu** on each platform's page.

After you stop the Sensu Core services, follow package removal instructions for your platform to uninstall the Sensu Core package.

Tune Sensu

This page describes tuning options that may help restore proper operation if you experience performance issues with your Sensu installation.

> **NOTE**: Before you tune your Sensu installation, read Troubleshoot Sensu, Hardware requirements, and Deployment architecture for Sensu. These pages describe common problems and solutions, planning and optimization considerations, and other recommendations that may resolve your issue without tuning adjustments.

Latency tolerances for etcd

If you use embedded etcd for storage, you might notice high network or storage latency.

To make etcd more latency-tolerant, increase the values for the etcd election timeout and etcd heartbeat interval backend configuration flags. For example, you might increase `etcd-election-timeout` from 100 to 500 and `etcd-heartbeat-interval` from 1000 to 5000.

Read the etcd tuning documentation for etcd-specific tuning best practices.

Advanced backend configuration options for etcd

The backend reference describes other advanced configuration flags in addition to etcd election timeout and heartbeat interval.

Adjust these values with caution. Improper adjustment can increase memory and CPU usage or result in a non-functioning Sensu instance.

Input/output operations per second (IOPS)

The speed with which write operations can be completed is important to Sensu cluster performance and health. Make sure to provision Sensu backend infrastructure to provide sustained input/output

operations per second (IOPS) appropriate for the rate of observability events the system will be required to process.

Read Backend recommended configuration and Hardware sizing for details.

PostgreSQL settings

The datastore reference lists the PostgreSQL configuration parameters and settings we recommend as a starting point for your `postgresql.conf` file. Adjust the parameters and settings as needed based on your hardware and performance observations.

Read the PostgreSQL parameters documentation for information about setting parameters.

Splay and proxy check scheduling

Adjust the `splay` and `splay_coverage` check attributes to tune proxy check executions across an interval. Read Fine-tune proxy check scheduling with splay for an example.

Tokens and resource re-use

Tokens are placeholders in a check, hook, or dynamic runtime asset definition that the agent replaces with entity information before execution. You can use tokens to fine-tune check, hook, and asset attributes on a per-entity level while reusing resource definitions.

Read the tokens reference for token syntax and examples.

Occurrences and alert fatigue

Use the `occurrences` and `occurrences_watermark` event attributes in event filters to tune incident notifications and reduce alert fatigue.

Troubleshoot Sensu

Service logging

Logs produced by Sensu services (sensu-backend and sensu-agent) are often the best place to start when troubleshooting a variety of issues.

Log levels

Each log message is associated with a log level that indicates the relative severity of the event being logged:

Log level	Description
panic	Severe errors that cause the service to shut down in an unexpected state
fatal	Fatal errors that cause the service to shut down (status 0)
error	Non-fatal service error messages
warn	Warning messages that indicate potential issues
info	Information messages that represent service actions
debug	Detailed service operation messages to help troubleshoot issues
trace	Confirmation messages about whether a rule authorized a request

You can configure these log levels by specifying the desired log level as the value of `log-level` in the service configuration file (`agent.yml` or `backend.yml`) or as an argument to the `--log-level` command line flag:

```
sensu-agent start --log-level debug
```

You must restart the service after you change log levels via configuration files or command line arguments. For help with restarting a service, read the <u>agent reference</u> or <u>backend reference</u>.

Increment log level verbosity

To increment the log level verbosity at runtime for the backend, run:

```
kill -s SIGUSR1 $(pidof sensu-backend)
```

To increment the log level verbosity at runtime for the agent, run:

```
kill -s SIGUSR1 $(pidof sensu-agent)
```

When you increment the log at the trace level (the most verbose log level), the log will wrap around to the error level.

Log file locations

Linux

Sensu services print <u>structured log messages</u> to standard output. To capture these log messages to disk or another logging facility, Sensu services use capabilities provided by the underlying operating system's service management. For example, logs are sent to the journald when systemd is the service manager, whereas log messages are redirected to `/var/log/sensu` when running under sysv init schemes. If you are running systemd as your service manager and would rather have logs written to `/var/log/sensu/`, read <u>forwarding logs from journald to syslog</u>.

For journald targets, use these commands to follow the logs. Replace the `<service>` variable with the name of the desired service (for example, `backend` or `agent`).

SHELL

```
journalctl --follow --unit sensu-<service>
```

SHELL

```
journalctl --follow --unit sensu-<service>
```

SHELL

```
journalctl --follow --unit sensu-<service>
```

For log file targets, use these commands to follow the logs. Replace the `<service>` variable with the name of the desired service (for example, `backend` or `agent`).

SHELL

```
tail --follow /var/log/sensu/sensu-<service>
```

SHELL

```
tail --follow /var/log/sensu/sensu-<service>
```

SHELL

```
tail --follow /var/log/sensu/sensu-<service>
```

> **NOTE**: Platform versions are listed for reference only and do not supersede the documented _supported platforms_.

Narrow your search to a specific timeframe

Use the `journald` keyword `since` to refine the basic `journalctl` commands and narrow your search by timeframe.

Retrieve all the logs for sensu-backend since yesterday:

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

Retrieve all the logs for sensu-agent since a specific time:

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

Retrieve all the logs for sensu-backend for a specific date range:

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

Logging edge cases

If a Sensu service experiences a panic crash, the service may seem to start and stop without producing any output in journalctl. This is due to a bug in systemd.

In these cases, try using the `_COMM` variable instead of the `-u` flag to access additional log entries:

```
journalctl _COMM=sensu-backend.service --since yesterday
```

Windows

The Sensu agent stores service logs to the location specified by the `log-file` configuration 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, read the agent reference. You can also view agent events using the Windows Event Viewer, under Windows Logs, as events with source SensuAgent.

If you're running a binary-only distribution of the Sensu agent for Windows, you can follow the service log printed to standard output using this command:

```
Get-Content -  Path "C:\scripts\test.txt" -Wait
```

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, read the 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 receive a logged error like those listed here if there is a permission issue with either the sensu-backend or the sensu-agent:

```
{"component":"agent","error":"open /var/cache/sensu/sensu-agent/assets.db:
permission denied","level":"fatal","msg":"error executing sensu-agent","time":"2019-
02-21T22:01:04Z"}
{"component":"backend","level":"fatal","msg":"error starting etcd: mkdir
/var/lib/sensu: permission denied","time":"2019-03-05T20:24:01Z"}
```

Use a recursive `chown` to resolve permission issues with the sensu-backend:

```
sudo chown -R sensu:sensu /var/cache/sensu/sensu-backend
```

or the sensu-agent:

```
sudo chown -R sensu:sensu /var/cache/sensu/sensu-agent
```

Handlers and event filters

Whether implementing new workflows or modifying existing workflows, you may need to troubleshoot various stages of the event pipeline.

Create an agent API test event

In many cases, generating events using the agent API will save you time and effort over modifying existing check configurations.

Here's an example that uses cURL with the API of a local sensu-agent process to generate test-event check results:

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "test-event"
    },
    "status": 2,
    "output": "this is a test event targeting the email_ops handler",
    "handlers": [ "email_ops" ]
  }
}' \
http://127.0.0.1:3031/events
```

Use a debug handler

It may also be helpful To review the complete event object being passed to your workflows.We recommend using a debug handler like this one to write an event to disk as JSON data:

YML

```yaml
---
type: Handler
api_version: core/v2
metadata:
  name: debug
spec:
  type: pipe
  command: cat > /var/log/sensu/debug-event.json
  timeout: 2
```

JSON

```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:

```bash
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "test-event"
    },
    "status": 2,
    "output": "this is a test event targeting the email_ops handler",
    "handlers": [ "email_ops", "debug" ]
  }
```

```
}' \
http://127.0.0.1:3031/events
```

The observability event data should be written to `/var/log/sensu/debug-event.json` for inspection.
The contents of this file will be overwritten by every event sent to the `debug` handler.

> **NOTE**: When multiple Sensu backends are configured in a cluster, event processing is distributed
> across all members. You may need to check the filesystem of each Sensu backend to locate the
> debug output for your test event.

Manually execute a handler

If you are not receiving events via a handler even though a check is generating events as expected,
follow these steps to manually execute the handler and confirm whether the handler is working
properly.

1. List all events:

    ```
    sensuctl event list
    ```

    Choose an event from the list to use for troubleshooting and note the event's check and entity
    names.

2. Navigate to the `/var/cache/sensu/sensu-backend/` directory:

    ```
    cd /var/cache/sensu/sensu-backend/
    ```

3. Run `ls` to list the contents of the `/var/cache/sensu/sensu-backend/` directory. In the list,
    identify the handler's dynamic runtime asset SHA.

    > **NOTE**: If the list includes more than one SHA, run `sensuctl asset list`. In the response,
    > the Hash column contains the first seven characters for each asset build's SHA. Note the
    > hash for your build of the handler asset and compare it with the SHAs listed in the
    > `/var/cache/sensu/sensu-backend/` directory to find the correct handler asset SHA.

4. Navigate to the `bin` directory for the handler asset SHA.Before you run the command below, replace `<handler_asset_sha>` with the SHA you identified in the previous step.

```
cd <handler_asset_sha>/bin
```

5. Run the command to manually execute the handler.Before you run the command below, replace the following text:

   - ⌐ `<entity_name>` : Replace with the entity name for the event you are using to troubleshoot.

   - ⌐ `<check_name>` : Replace with the check name for the event you are using to troubleshoot.

   - ⌐ `<handler_command>` : Replace with the `command` value for the handler you are troubleshooting.

```
sensuctl event info <entity_name> <check_name> --format json |
./<handler_command>
```

If your handler is working properly, you will receive an alert for the event via the handler.The response for your manual execution command will also include a message to confirm notification was sent.In this case, your Sensu pipeline is not causing the problem with missing events.

If you do not receive an alert for the event, the handler is not working properly.In this case, the manual execution response will include the message `Error executing <handler_asset_name>:` followed by a description of the specific error to help you correct the problem.

Dynamic runtime assets

Use the information in this section to troubleshoot error messages related to dynamic runtime assets.

Incorrect asset filter

Dynamic runtime asset filters allow you to scope an asset to a particular operating system or architecture.For an example, read the asset reference.An improperly applied asset filter can prevent the

asset from being downloaded by the desired entity and result in error messages both on the agent and the backend illustrating that the command was not found:

Agent log entry

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

Backend event

YML

```
---
timestamp: 1568148292
check:
  command: check-disk-space
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  publish: true
  runtime_assets:
  - sensu-plugins-disk-checks
  subscriptions:
  - caching_servers
  proxy_entity_name: ''
  check_hooks:
  stdin: false
  subdue:
  ttl: 0
  timeout: 0
  round_robin: false
```

```yaml
    duration: 0.001795508
    executed: 1568148292
    history:
    - status: 127
      executed: 1568148092
    issued: 1568148292
    output: 'sh: check-disk-space: command not found'
    state: failing
    status: 127
    total_state_change: 0
    last_ok: 0
    occurrences: 645
    occurrences_watermark: 645
    output_metric_format: ''
    output_metric_handlers:
    output_metric_tags:
    env_vars:
    metadata:
      name: failing-disk-check
      namespace: default
  metadata:
    namespace: default
```

JSON

```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,
    "output_metric_tags": null,
    "env_vars": null,
    "metadata": {
      "name": "failing-disk-check",
      "namespace": "default"
    }
  },
  "metadata": {
    "namespace": "default"
  }
}
```

If you receive a message like this, review your asset definition — it means that the entity wasn't able to download the required asset due to asset filter restrictions. To review the filters for an asset, use the sensuctl `asset info` command with a `--format` flag:

SHELL

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

SHELL

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

Conflating operating systems with families

A common asset filter issue is conflating operating systems with the family they're a part of. For example, although Ubuntu is part of the Debian family of Linux distributions, Ubuntu is not the same as Debian. A practical example might be:

YML

```
filters:
- entity.system.platform == 'debian'
- entity.system.arch == 'amd64'
```

JSON

```
{
  "filters": [
    "entity.system.platform == 'debian'",
    "entity.system.arch == 'amd64'"
  ]
}
```

This would not allow an Ubuntu system to run the asset.

Instead, the asset filter should look like this:

YML

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

JSON

```json
{
  "filters": [
    "entity.system.platform_family == 'debian'",
    "entity.system.arch == 'amd64'"
  ]
}
```

or

YML

```yaml
filters:
- entity.system.platform == 'ubuntu'
- entity.system.arch == 'amd64'
```

JSON

```json
{
  "filters": [
    "entity.system.platform == 'ubuntu'",
    "entity.system.arch == 'amd64'"
  ]
}
```

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

Running the agent on an unsupported Linux platform

If you run the Sensu agent on an unsupported Linux platform, the agent might fail to correctly identify your version of Linux and could download the wrong version of an asset.

This issue affects Linux distributions that do not include the `lsb_release` package in their default installations. In this case, `gopsutil` may try to open `/etc/lsb_release` or try to run `/usr/bin/lsb_release` to find system information, including Linux version. Since the `lsb_release` package is not installed, the agent will not be able to discover the Linux version as expected.

To resolve this problem, install the `lsb_release` package for your Linux distribution.

Checksum mismatch

When a downloaded dynamic runtime asset checksum does not match the checksum specified in the asset definition, the agent logs a message similar to this example:

```
{
  "asset": "check-disk-space",
  "component": "asset-manager",
  "entity": "sensu-centos",
  "filters": [
    "true == false"
  ],
  "level": "debug",
  "msg": "error getting assets for check: could not validate downloaded asset
$ASSET_NAME (X.X MB): sha512 of downloaded asset
(6b73p32XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX) does not match specified sha512 in
asset definition
(e6b7c8eXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX)",
  "time": "2020-09-12T18:28:05Z"
}
```

To correct this issue, first confirm that the URL in the asset definition is valid.Manually download the asset with a cURL or wget command and make sure that the downloaded file is a valid `tar.gz` file with the contents you expect.

If the downloaded `tar.gz` file contents are correct, use the `sha512sum` command to calculate the asset checksum and manually confirm that the checksum in the downloaded asset definition is correct.

If the checksum in the downloaded asset definition is correct, confirm that there is enough space available in `/tmp` to download the asset.On Linux systems, the Sensu agent downloads assets into `/tmp`.The log error message specifies the size of the asset artifact in parentheses after the asset name.If space in `/tmp` is insufficient, asset downloads will be truncated and the checksum will not be validated.

Etcd clusters

Some issues require you to investigate the state of the etcd cluster or data stored within etcd. In these cases, we suggest using the etcdctl tool to query and manage the etcd database.

Sensu's supported packages do not include the etcdctl executable, so you must get it from a compatible etcd release.

Configure etcdctl environment variables

To use etcdctl to investigate etcd cluster and data storage issues, first run these commands to configure etcdctl environment variables:

```
export ETCDCTL_API=3
export ETCDCTL_CACERT=/etc/sensu/ca.pem
export
ETCDCTL_ENDPOINTS="https://backend01:2379,https://backend02:2379,https://backend03:2
379"
```

If your etcd uses client certificate authentication, run these commands too:

```
export ETCDCTL_CERT=/etc/sensu/cert.pem
export ETCDCTL_KEY=/etc/sensu/key.pem
```

View cluster status and alarms

Use the commands listed here to retrieve etcd cluster status and list and clear alarms.

To retrieve etcd cluster status:

```
etcdctl endpoint status
```

To retrieve a list of etcd alarms:

```
etcdctl alarm list
```

To clear etcd alarms:

```
etcdctl alarm dearm
```

Restore a cluster with an oversized database

The etcd default maximum database size is 2 GB.If you suspect your etcd database exceeds the maximum size, run this command to confirm cluster size:

```
etcdctl endpoint status
```

The response will list the current cluster status and database size:

```
https://backend01:2379, 88db026f7feb72b4, 3.3.22, 2.1GB, false, 144, 18619245
https://backend02:2379, e98ad7a888d16bd6, 3.3.22, 2.1GB, true, 144, 18619245
https://backend03:2379, bc4e39432cbb36d, 3.3.22, 2.1GB, false, 144, 18619245
```

To restore an etcd cluster with a database size that exceeds 2 GB:

1. Get the current revision number:

   ```
   etcdctl endpoint status --write-out="json" | egrep -o '"revision":[0-9]*' |
   egrep -o '[0-9].*'
   ```

2. Compact to revision and substitute the current revision for `$rev`:

   ```
   etcdctl compact $rev
   ```

3. Defragment to free up space:

```
etcdctl defrag
```

4. Confirm that the cluster is restored:

```
etcdctl endpoint status
```

The response should list the current cluster status and database size:

```
https://backend01:2379, 88db026f7feb72b4, 3.3.22, 1.0 MB, false, 144, 18619245
https://backend02:2379, e98ad7a888d16bd6, 3.3.22, 1.0 MB, true, 144, 18619245
https://backend03:2379, bc4e39432cbb36d, 3.3.22, 1.0 MB, false, 144, 18619245
```

Remove and redeploy a cluster

PRO TIP: Use etcd snapshots to keep a backup so that you can restore your Sensu resources if you have to redeploy your cluster. For extra reassurance, take regular etcd snapshots and make regular backups with sensuctl dump in addition to etcd's running snapshots.

If you wait until cluster nodes are failing, it may not be possible to make a backup. For example, in a three-node cluster, if one node fails, you will still be able to make a backup. If two nodes fail, the whole cluster will be down and you will not be able to create a snapshot or run sensuctl dump.

You may need to completely remove a cluster and redeploy it in cases such as:

-   Failure to reach consensus after losing more than `(N-1)/2` cluster members
-   Etcd configuration issues
-   Etcd corruption, perhaps from disk filling
-   Unrecoverable hardware failure

To remove and redeploy a cluster:

1. Open a terminal window for each cluster member.

2. Stop each cluster member backend:

```
systemctl stop sensu-backend
```

3. Confirm that each backend stopped:

```
systemctl status sensu-backend
```

For each backend, the response should begin with the following lines:

```
• sensu-backend.service - The Sensu Backend service.
Loaded: loaded (/usr/lib/systemd/system/sensu-backend.service; disabled; vendor preset: disabled)
Active: inactive (dead)
```

4. Delete the etcd directories for each cluster member:

```
rm -rf /var/lib/sensu/sensu-backend/etcd/
```

5. Follow the Sensu backend configuration instructions to reconfigure a new cluster.

6. Initialize a backend to specify admin credentials:

```
sensu-backend init --interactive
```

When you receive prompts for your username and password, replace `<YOUR_USERNAME>` and `<YOUR_PASSWORD>` with the administrator username and password you want to use for the cluster members:

```
Admin Username: <YOUR_USERNAME>
```

```
    Admin Password: <YOUR_PASSWORD>
```

7. Follow the etcd restore process or use sensuctl create to restore your cluster from a snapshot
   or backup.

Datastore performance

In a default deployment, Sensu uses etcd datastore for both configuration and state. As the number of
checks and entities in your Sensu installation increases, so does the volume of read and write requests
to etcd database.

One trade-off in etcd's design is its sensitivity to disk and CPU latency. When certain latency tolerances
are regularly exceeded, failures will cascade. Sensu will attempt to recover from these conditions when
it can, but this may not be successful.

To maximize Sensu Go performance, we recommend that you:

- ¬ Follow our recommended backend hardware configuration.

- ¬ Implement documented etcd system tuning practices.

- ¬ Benchmark your etcd storage volume to establish baseline IOPS for your system.

- ¬ Scale event storage using PostgreSQL to reduce the overall volume of etcd transactions.

As your Sensu deployments grow, preventing issues associated with poor datastore performance
relies on ongoing collection and review of Sensu time-series performance metrics.

Symptoms of poor performance

At the default "warn" log level, you may receive messages like these from your Sensu backend:

```
{"component":"etcd","level":"warning","msg":"read-only range request
\"key:\\\"/sensu.io/handlers/default/keepalive\\\" limit:1 \" with result
\"range_response_count:0 size:6\" took too long (169.767546ms) to
execute","pkg":"etcdserver","time":"..."}
```

The above message indicates that a database query ("read-only range request") exceeded a 100-

millisecond threshold hard-coded into etcd. Messages like these are helpful because they can alert you to a trend, but these occasional warnings don't necessarily indicate a problem with Sensu. For example, you may receive this message if you provision attached storage but do not mount it to the etcd data directory.

However, a trend of increasingly long-running database transactions will eventually lead to decreased reliability. You may experience symptoms of these conditions as inconsistent check execution behavior or configuration updates that are not applied as expected.

As the etcd tuning documentation states:

> An etcd cluster is very sensitive to disk latencies. Since etcd must persist proposals to its log, disk activity from other processes may cause long fsync latencies. […] etcd may miss heartbeats, causing request timeouts and temporary leader loss.

When Sensu's etcd component doesn't recieve sufficient CPU cycles or its file system can't sustain a sufficient number of IOPS, transactions will begin to timeout, leading to cascading failures.

A message like this indicates that syncing the etcd database to disk exceeded another threshold:

```
{"component":"etcd","level":"warning","msg":"sync duration of 1.031759056s, expected
less than 1s","pkg":"wal","time":"..."}
```

These subsequent "retrying of unary invoker failed" messages indicate failing requests to etcd:

```
{"level":"warn","ts":"...","caller":"clientv3/retry_interceptor.go:62","msg":"retryi
ng of unary invoker failed","target":"endpoint://client-6f6bfc7e-cf31-4498-a564-
78d6b7b3a44e/localhost:2379","attempt":0,"error":"rpc error: code = Canceled desc =
context canceled"}
```

On busy systems you may also receive output like "message repeated 5 times" indicating that failing requests were retried multiple times.

In many cases, the backend service detects and attempts to recover from errors like these, so you may receive a message like this:

```
{"component":"backend","error":"error from keepalived: internal error: etcdserver:
```

```
request timed out","level":"error","msg":"backend stopped working and is
restarting","time":"..."}
```

This may result in a crash loop that is difficult to recover from.You may observe that the Sensu backend process continues running but is not listening for connections on the agent WebSocket, API, or web UI ports.The backend will stop listening on those ports when the etcd database is unavailable.

License reference

Activate your commercial license

If you haven't already, <u>install the backend, agent, and sensuctl</u> and <u>configure sensuctl</u>.

Log in to your Sensu account at <u>account.sensu.io</u> 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

Save your license to a file such as `sensu_license.yml` or `sensu_license.json` .With the license file downloaded and saved to a file, you can activate your license with sensuctl or the <u>license API</u>.

To activate your license with sensuctl:

SHELL

```
sensuctl create --file sensu_license.yml
```

SHELL

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

Use sensuctl to view your license details at any time:

```
sensuctl license info
```

For an active license, the response should be similar to this example:

```
=== You are currently using 10/100 total entities, 5/50 agent entities, and 5/50
proxy entities
Account Name: Training Team - Sensu
Account ID:   123
Plan:         managed
Version:      1
Features:     all
Issuer:       Sensu, Inc.
Issued:       2020-02-15 15:01:44 -0500 -0500
Valid:        true
Valid Until:  2021-02-15 00:00:00 -0800 -0800
```

This response means you do not have an active license:

```
Error: not found
```

Entity limit

Your commercial license may include the entity limit and entity class limits tied to your Sensu licensing package.Contact Sensu to upgrade your commercial license.

Your Sensu license may include two types of entity limits:

- Entity limit: the maximum number of entities of all classes your license includes.Both agent and proxy entities count toward the overall entity limit.

- Entity class limits: the maximum number of a specific class of entities (for example, agent or proxy) that your license includes.

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

View entity count and entity limit

Your current entity count and entity limit are included in the `sensuctl license info` response.

In tabular format, the entity count and limit are included in the response title.To view license info in tabular format, run:

```
sensuctl license info --format tabular
```

The response in tabular format should be similar to this example:

```
=== You are currently using 10/100 total entities, 5/50 agent entities, and 5/50
proxy entities
Account Name: Training Team - Sensu
Account ID:   123
Plan:         managed
```

```
Version:     1
Features:    all
Issuer:      Sensu, Inc.
Issued:      2020-02-15 15:01:44 -0500 -0500
Valid:       true
Valid Until: 2021-02-15 00:00:00 -0800 -0800
```

If you have an unlimited entity count, the `sensuctl license info` response title will still include a current count for each type of entity you are using.For example:

```
=== You are currently using 10/unlimited total entities, 5/unlimited agent entities,
and 5/unlimited proxy entities
```

To view license details in YAML or JSON, run:

SHELL

```
sensuctl license info --format yaml
```

SHELL

```
sensuctl license info --format wrapped-json
```

In YAML and JSON formats, the entity count and limit are included as labels:

YML

```
type: LicenseFile
api_version: licensing/v2
metadata:
  labels:
    sensu.io/entity-count: "10"
    sensu.io/entity-limit: "100"
spec:
  license:
    version: 1
    issue: Sensu, Inc.
    accountName: Training Team - Sensu
```

```
[...]
```

JSON

```json
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issue": "Sensu, Inc.",
      "accountName": "Training Team - Sensu"
    },
    "...": "..."
  }
}
```

You can also find your current entity count and limit in the response headers for any `/api/core` or `/api/enterprise` API request. For example:

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/entities -v -H
"Authorization: Key $SENSU_API_KEY"
```

The response headers will include your current entity count and limit:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Entity-Count: 10
Sensu-Entity-Limit: 100
```

License expiration

To view your commercial license expiration date, log in to your Sensu account.

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-tier dynamic runtime assets

- ⌐ Use dynamic runtime assets to install plugins

- ⌐ Contact Sensu support

- ⌐ Contact Sensu sales

Monitor Sensu

Use the guides and references in the Monitor Sensu category to successfully monitor your Sensu installation.

Log Sensu services and monitor with Sensu

Learn how to log Sensu services with systemd, including adding log forwarding from journald to syslog, using rsyslog to write logging data to disk, and setting up log rotation.

Read Monitor Sensu with Sensu to monitor the Sensu backend with another Sensu backend or cluster: use a secondary Sensu instance to notify you when your primary Sensu instance is down (and vice versa).

Retrieve cluster health data

The health reference explains how to use Sensu's health API to ensure your backend is up and running and check the health of your etcd cluster members and PostgreSQL datastore resources. Learn how to read the JSON response for health API requests by reviewing examples of responses for clusters with healthy and unhealthy members and the response specification.

Learn about Tessen

The Tessen reference explains the Sensu call-home service, which is enabled by default on Sensu backends and required for licensed Sensu instances. We rely on anonymized Tessen data to understand how Sensu is being used and make informed decisions about product improvements.

Log Sensu services with systemd

By default, systems where systemd is the service manager do not write logs to `/var/log/sensu/` for the `sensu-agent` and the `sensu-backend` services. This guide explains how to add log forwarding from journald to syslog, have rsyslog write logging data to disk, and set up log rotation of the newly created log files.

Configure journald

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

```
ForwardToSyslog=yes
```

Configure rsyslog

Next, set up rsyslog to write the logging data received from journald to `/var/log/sensu/servicename.log`. In this example, the `sensu-backend` and `sensu-agent` logging data is sent to individual files named after the service. The `sensu-backend` is not required if you're only setting up log forwarding for the `sensu-agent` service.

> **NOTE**: Use a `conf` file name that will ensure the file is loaded before the default file in `/etc/rsyslog.d/`, which uses 50. This example uses `40-sensu-backend.conf` and `40-sensu-agent.conf` for this reason.

1. For the sensu-backend service, in /etc/rsyslog.d/40-sensu-backend.conf, add:

   ```
   if $programname == 'sensu-backend' then {
           /var/log/sensu/sensu-backend.log
           & stop
   }
   ```

2. For the sensu-agent service, in /etc/rsyslog.d/40-sensu-agent.conf, add:

```
if $programname == 'sensu-agent' then {
        /var/log/sensu/sensu-agent.log
        & stop
}
```

3. **On Ubuntu systems**, run `chown -R syslog:adm /var/log/sensu` so syslog can write to that directory.

4. Restart journald:

```
systemctl restart systemd-journald
```

5. Restart rsyslog to apply the new configuration:

```
systemctl restart rsyslog
```

> *NOTE: Sensu log messages include the Sensu <u>log level</u> as part of the log data. Users with rsyslog expertise may be able to extract the log level from Sensu log messages and use rsyslog processing capabilities to separate the log messages into different files based on log level.*

Set up log rotation

Set up log rotation for newly created log files to ensure logging does not fill up your disk.

These examples rotate the log files `/var/log/sensu/sensu-agent.log` and `/var/log/sensu/sensu-backend.log` weekly, unless the size of 100M is reached first. The last seven rotated logs are kept and compressed, with the exception of the most recent log. After rotation, `rsyslog` is restarted to ensure logging is written to a new file and not the most recent rotated file.

1. In /etc/logrotate.d/sensu-agent.conf, add:

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

2. In /etc/logrotate.d/sensu-backend.conf, add:

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

Use the following command to find out what logrotate would do if it were executed now based on the above schedule and size threshold. The `-d` flag will output details, but it will not take action on the logs or execute the postrotate script:

```
logrotate -d /etc/logrotate.d/sensu.conf
```

Next steps

Sensu also offers logging of observability event data to a separate JSON log file as a commercial feature. Read the Sensu backend reference for more information about event logging.

Monitor Sensu with Sensu

This guide describes best practices and strategies for monitoring the Sensu backend with another Sensu backend or cluster.

To completely monitor Sensu (a Sensu backend with internal etcd and an agent), you will need at least one independent Sensu instance in addition to the primary instance you want to monitor. The second Sensu instance will ensure that you are notified when the primary is down and vice versa.

This guide requires Sensu plugins using dynamic runtime assets. For more information about using Sensu plugins, read Use dynamic runtime assets to install plugins.

> **NOTE**: Although this guide describes approaches for monitoring a single backend, these strategies are also useful for monitoring individual members of a backend cluster.
>
> This guide does not describe Sensu agent keepalive monitoring.

The checks in this guide monitoring the following ports and endpoints:

Port	Endpoint	Description
2379	`/health`	Etcd health endpoint. Provides health status for etcd nodes.
8080	`/health`	Sensu Go health endpoint. Provides health status for Sensu backends, as well as for PostgreSQL (when enabled).

Register dynamic runtime assets

To power the checks to monitor your Sensu backend, external etcd, and PostgreSQL instances, add the Sensu HTTP Plugin dynamic runtime asset. This asset includes the `check-http.rb` plugin, which your checks will rely on.

The Sensu assets packaged from Sensu HTTP Plugin are built against the Sensu Ruby runtime environment, so you also need to add the <u>Sensu Go Ruby Runtime Assets</u> dynamic runtime asset. The Ruby runtime asset delivers the Ruby executable and supporting libraries the check will need to run the `check-http.rb` plugin.

To do this, use the `check-http.rb` plugin from the <u>Sensu HTTP Plugin</u> dynamic runtime asset to query Sensu's <u>health API endpoint</u> with a check definition for your primary (Backend Alpha) and secondary (Backend Beta) backends:

To register the Sensu HTTP Plugin dynamic runtime asset, `sensu-plugins/sensu-plugins-http:6.0.0` , run:

```
sensuctl asset add sensu-plugins/sensu-plugins-http:6.0.0 -r sensu-plugins-http
```

The response will confirm that the asset was added:

```
fetching bonsai asset: sensu-plugins/sensu-plugins-http:6.0.0
added asset: sensu-plugins/sensu-plugins-http:6.0.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu-plugins-http"].
```

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

Then, use the following sensuctl command to register the Sensu Ruby Runtime dynamic runtime asset, `sensu/sensu-ruby-runtime:0.1.0` :

```
sensuctl asset add sensu/sensu-ruby-runtime:0.1.0 -r sensu-ruby-runtime
```

To confirm that all three dynamic runtime assets are ready to use, run:

```
sensuctl asset list
```

The response should list the `sensu-plugins-http` and `sensu-ruby-runtime` dynamic runtime assets:

```
    Name                          URL                              Hash
 _____

_____
_____ _____

  sensu-plugins-http   //assets.bonsai.sensu.io/.../sensu-plugins-http_6.0.0_debian9_linux_amd64.tar.gz      ed9c3c8

  sensu-plugins-http   //assets.bonsai.sensu.io/.../sensu-plugins-http_6.0.0_debian_linux_amd64.tar.gz       bfa025f

  ...

  sensu-ruby-runtime   //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-2.4.4_amzn2_linux_amd64.tar.gz   a83aaa5

  sensu-ruby-runtime   //assets.bonsai.sensu.io/.../sensu-ruby-runtime_0.1.0_ruby-2.4.4_amzn1_linux_amd64.tar.gz
7b504f0

  ...
```

Because plugins are published for multiple platforms, including Linux and Windows, the output will include multiple entries for each of the dynamic runtime assets.

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until they are needed for command execution.Read _the asset reference_ for more information about dynamic runtime asset builds.

Monitor your Sensu backend instances

Monitor the host running the `sensu-backend` _locally_ by a `sensu-agent` process for operating system checks and metrics.

For Sensu components that must be running for Sensu to create events, you should also monitor the `sensu-backend` remotely from an independent Sensu instance.This will allow you to monitor whether your Sensu event pipeline is working.

To do this, add checks that use the `check-http.rb` plugin from the Sensu HTTP Plugin dynamic runtime asset to query Sensu's health API endpoint for your primary (Backend Alpha) and secondary (Backend Beta) backends:

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  namespace: default
  name: check_beta_backend_health
spec:
  command: check-http.rb -u http://sensu-backend-beta:8080/health -n false
  subscriptions:
    - backend_alpha
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
    - sensu-ruby-runtime
    - sensu-plugins-http
```

YML

```yaml
---
type: CheckConfig
api_version: core/v2
metadata:
  namespace: default
  name: check_alpha_backend_health
spec:
  command: check-http.rb -u http://sensu-backend-beta:8080/health -n false
  subscriptions:
    - backend_beta
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
    - sensu-ruby-runtime
    - sensu-plugins-http
```

JSON

```json
{
```

```
    "type": "CheckConfig",
    "api_version": "core/v2",
    "metadata": {
      "namespace": "default",
      "name": "check_beta_backend_health"
    },
    "spec": {
      "command": "check-http.rb -u http://sensu-backend-beta:8080/health -n false",
      "subscriptions": [
        "backend_alpha"
      ],
      "interval": 10,
      "publish": true,
      "timeout": 10,
      "runtime_assets": [
        "sensu-ruby-runtime",
        "sensu-plugins-http"
      ]
    }
  }
```

JSON

```
  {
    "type": "CheckConfig",
    "api_version": "core/v2",
    "metadata": {
      "namespace": "default",
      "name": "check_alpha_backend_health"
    },
    "spec": {
      "command": "check-http.rb -u http://sensu-backend-beta:8080/health -n false",
      "subscriptions": [
        "backend_beta"
      ],
      "interval": 10,
      "publish": true,
      "timeout": 10,
      "runtime_assets": [
        "sensu-ruby-runtime",
        "sensu-plugins-http"
      ]
```

```
    }
  }
```

Monitor external etcd

If your Sensu Go deployment uses an external etcd cluster, you'll need to check the health of the respective etcd instances for your primary (Backend Alpha) and secondary (Backend Beta) backends.

This example uses the `check-http.rb` plugin from the <u>Sensu HTTP Plugin</u> dynamic runtime asset:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  namespace: default
  name: check_beta_etcd_health
spec:
  command: check-http.rb -u http://sensu-etcd-beta:2379/health -n false
  subscriptions:
  - backend_alpha
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
  - sensu-plugins/sensu-plugins-http
  - sensu/sensu-ruby-runtime
```

YML

```yml
---
type: CheckConfig
api_version: core/v2
```

```yaml
metadata:
  namespace: default
  name: check_alpha_etcd_health
spec:
  command: check-http.rb -u http://sensu-etcd-alpha:2379/health -n false
  subscriptions:
  - backend_beta
  interval: 10
  publish: true
  timeout: 10
  runtime_assets:
  - sensu-plugins/sensu-plugins-http
  - sensu/sensu-ruby-runtime
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "check_beta_etcd_health"
  },
  "spec": {
    "command": "check-http.rb -u http://sensu-etcd-beta:2379/health -n false",
    "subscriptions": [
      "backend_alpha"
    ],
    "interval": 10,
    "publish": true,
    "timeout": 10,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-http",
      "sensu/sensu-ruby-runtime"
    ]
  }
}
```

JSON

```json
{
```

```
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "check_alpha_etcd_health"
  },
  "spec": {
    "command": "check-http.rb -u http://sensu-etcd-alpha:2379/health -n false",
    "subscriptions": [
      "backend_beta"
    ],
    "interval": 10,
    "publish": true,
    "timeout": 10,
    "runtime_assets": [
      "sensu-plugins/sensu-plugins-http",
      "sensu/sensu-ruby-runtime"
    ]
  }
}
```

NOTE: *This example uses the Sensu HTTP Plugin and Sensu Go Ruby Runtime dynamic runtime assets. Follow Register dynamic runtime assets if you have not previously added these assets.*

Monitor PostgreSQL

COMMERCIAL FEATURE: *Access enterprise-scale PostgreSQL event storage in the packaged Sensu Go distribution. For more information, read Get started with commercial features.*

Larger Sensu deployments may use PostgreSQL as an alternative datastore to process larger numbers of events. The connection to PostgreSQL is exposed on Sensu's `/health` endpoint and will look like this example:

```
{
  "Alarms": null,
  "ClusterHealth": [{
```

```json
    "MemberID": 3470366781180380542,
    "MemberIDHex": "302938336092857e",
    "Name": "sensu00",
    "Err": "",
    "Healthy": true
  }, {
    "MemberID": 15883454222313069303,
    "MemberIDHex": "dc6d5d7607261af7",
    "Name": "sensu01",
    "Err": "",
    "Healthy": true
  }, {
    "MemberID": 11377294497886211005,
    "MemberIDHex": "9de44510fb838bbd",
    "Name": "sensu02",
    "Err": "",
    "Healthy": true
  }],
  "Header": {
    "cluster_id": 13239446193995634903,
    "member_id": 3470366781180380542,
    "raft_term": 1549
  },
  "PostgresHealth": [{
    "Name": "sensu_postgres",
    "Active": true,
    "Healthy": true
  }]
}
```

To monitor PostgreSQL's health from Sensu's perspective, use a check like this example, which uses the `check-http.rb` plugin from the Sensu HTTP Plugin dynamic runtime asset:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  created_by: admin
  labels:
```

```
      sensu.io/managed_by: sensuctl
  name: check-postgres-health
  namespace: default
spec:
  check_hooks: null
  command: check-http-json.rb -u https://sensu.example.com:8080/health --key
PostgresHealth[0].Healthy
    --value true
  env_vars: null
  handlers: []
  high_flap_threshold: 0
  interval: 10
  low_flap_threshold: 0
  output_metric_format: ""
  output_metric_handlers: null
  output_metric_tags: null
  proxy_entity_name: ""
  publish: true
  round_robin: true
  runtime_assets:
  - sensu-plugins/sensu-plugins-http
  - sensu/sensu-ruby-runtime
  secrets: null
  stdin: false
  subdue: null
  subscriptions:
  - backends
  timeout: 0
  ttl: 0
```

JSON

```
{
  "command": "check-http-json.rb -u https://sensu.example.com:8080/health --key
PostgresHealth[0].Healthy --value true",
  "handlers": [],
  "high_flap_threshold": 0,
  "interval": 10,
  "low_flap_threshold": 0,
  "publish": true,
  "runtime_assets": [
    "sensu-plugins/sensu-plugins-http",
```

```
      "sensu/sensu-ruby-runtime"
    ],
    "subscriptions": [
      "backends"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": true,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "output_metric_tags": null,
    "env_vars": null,
    "metadata": {
      "name": "check-postgres-health",
      "namespace": "default",
      "labels": {
        "sensu.io/managed_by": "sensuctl"
      },
      "created_by": "admin"
    },
    "secrets": null
  }
```

> **NOTE**: This example uses the _Sensu HTTP Plugin_ and _Sensu Go Ruby Runtime_ dynamic runtime assets. Follow _Register dynamic runtime assets_ if you have not previously added these assets.

A successful PostgreSQL health check result will be similar to this example:

Timeline

✓

01:25   01:30   Now
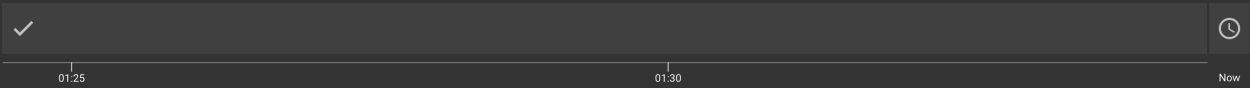
Check Result

Status	✓ success (0)	Check	check-postgres-health
Total State Change	0%	Entity	▪ ▪
Last OK	36 minutes ago	Issued at	Jul 6, 12:58 PM
Occurrences	10,473	Ran at	12:58 PM for 160ms
Max Occurrences	10,473		

```
CheckJson OK: key has expected value: 'PostgresHealth[0].Healthy' equals 'true'
```

CHECK CONFIGURATION SUMMARY ⌄

Health reference

Use Sensu's health API to make sure your backend is up and running and check the health of your etcd cluster members and PostgreSQL datastore resources.

A request to the health endpoint retrieves a JSON map with health data for your Sensu instance.

Healthy cluster example

In this example, all cluster members are healthy.

```
curl -X GET \
http://127.0.0.1:8080/health

HTTP/1.1 200 OK
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 9861478486968594000,
      "MemberIDHex": "88db026f7feb72b4",
      "Name": "backend01",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 16828500076473182000,
      "MemberIDHex": "e98ad7a888d16bd6",
      "Name": "backend02",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 848052855499371400,
      "MemberIDHex": "bc4e39432cbb36d",
      "Name": "backend03",
```

```
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 17701109828877156000,
    "member_id": 16828500076473182000,
    "raft_term": 42
  }
},
  "PostgresHealth": [
    {
      "Name": "my-first-postgres",
      "Active": true,
      "Healthy": true
    },
    {
      "Name": "my-other-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```

Unhealthy cluster member example

In this example, one cluster member is unhealthy: it cannot communicate with the other cluster members.

```
curl -X GET \
http://127.0.0.1:8080/health

HTTP/1.1 200 OK
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 9861478486968594000,
```

```json
      "MemberIDHex": "88db026f7feb72b4",
      "Name": "backend01",
      "Err": "context deadline exceeded",
      "Healthy": false
    },
    {
      "MemberID": 16828500076473182000,
      "MemberIDHex": "e98ad7a888d16bd6",
      "Name": "backend02",
      "Err": "",
      "Healthy": true
    },
    {
      "MemberID": 848052855499371400,
      "MemberIDHex": "bc4e39432cbb36d",
      "Name": "backend03",
      "Err": "",
      "Healthy": true
    }
  ],
  "Header": {
    "cluster_id": 17701109828877156000,
    "member_id": 16828500076473182000,
    "raft_term": 42
  }
},
  "PostgresHealth": [
    {
      "Name": "my-first-postgres",
      "Active": true,
      "Healthy": true
    },
    {
      "Name": "my-other-postgres",
      "Active": false,
      "Healthy": false
    }
  ]
}
```

NOTE: The HTTP response codes for the health endpoint indicate whether your request reached

Sensu rather than the health of your Sensu instance. In this example, even though the cluster is unhealthy, the request itself reached Sensu, so the response code is `200 OK` *. To determine the health of your Sensu instance, you must process the JSON response body. The health specification describes each attribute in the response body.*

Health specification

Top-level attributes

Alarms	
description	Top-level attribute that lists all active etcd alarms.
required	true
type	String
example	`"Alarms": null`

ClusterHealth	
description	Top-level attribute that includes health status information for every etcd cluster member.
required	true
type	Map of key-value pairs
| example | ```
"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	<pre>"Header": {
	    "cluster_id": 4255616344056076734,
	    "member_id": 2882886652148554927,
	    "raft_term": 26
	  }</pre>

PostgresHealth

description	Top-level map that includes health information for PostgreSQL resources. If your Sensu instance is not configured to use a PostgreSQL datastore, the health payload will not include `PostgresHealth`.
type	Map of key-value pairs

example	<pre>"PostgresHealth": [
	    {
	      "Name": "postgres-test",
	      "Active": false,
	      "Healthy": false
	    },
	    {
	      "Name": "postgres",
	      "Active": true,
	      "Healthy": true</pre>

```
      }
   ]
```

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`

PostgresHealth attributes

Name

description	The PostgreSQL configuration resource. Sensu retrieves the `Name` from datastore metadata.
required	true

type	String

example	
	``` "Name": "postgres" ```

Active

description	`true` if the datastore is configured to use the PostgreSQL configuration. Otherwise, `false`.
required	true
type	Boolean
default	`false`

example	
	``` "Active": true ```

Healthy

description	`true` if the PostgreSQL datastore is connected and can query the events table. Otherwise, `false`.
required	true
type	Boolean
default	`false`

example	
	``` "Healthy": true ```

Tessen reference

Tessen is the Sensu call-home service. It is enabled by default on Sensu backends. Tessen sends anonymized data about Sensu instances to Sensu Inc., including the version, cluster size, number of events processed, and number of resources created (like checks and handlers). We rely on Tessen data to understand how Sensu is being used and make informed decisions about product improvements. Read Announcing Tessen, the Sensu call-home service to learn more about Tessen.

All data submissions are logged for complete transparency at the `info` log level and transmitted over HTTPS. Read Troubleshoot Sensu to set the Sensu backend log level and view logs.

Configure Tessen

You can use the Tessen API and sensuctl to view your Tessen configuration. If you are using an unlicensed Sensu instances, you can also use the Tessen API and sensuctl to opt in or opt out of Tessen.

> **NOTE**: Tessen is enabled by default on Sensu backends and required for licensed Sensu instances. If you have a licensed instance and want to opt out of Tessen, contact your account manager.

To manage Tessen configuration for your unlicensed instance with sensuctl, configure sensuctl as the default `admin` user.

To view Tessen status:

```
sensuctl tessen info
```

To opt out of Tessen:

```
sensuctl tessen opt-out
```

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	**YML**
	
	```type: TessenConfig```
	
	JSON
	
	```{ "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

YML
```
api_version: core/v2
```

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

spec

description	Top-level map that includes Tessen configuration spec attributes.
required	Required for Tessen configuration in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

example

YML
```
spec:
  opt_out: false
```

JSON

```
{
  "spec": {
    "opt_out": false
  }
}
```

Spec attributes

opt_out	
description	`true` to opt out of Tessen. Otherwise, `false`. Tessen is enabled by default on Sensu backends and required for licensed Sensu instances.
required	true
type	Boolean
default	`false`

example	**YML**
	
	```opt_out: false```
	
	JSON
	
	``{
	  "opt_out": false
	}```

Tessen configuration example

This example is in `wrapped-json` format for use with `sensuctl create`. To manage Tessen for unlicensed Sensu instances with the Tessen API, use non-wrapped `json` format as shown in the API docs.

YML

```yml
---
type: TessenConfig
api_version: core/v2
spec:
  opt_out: false
```

JSON

```json
{
  "type": "TessenConfig",
  "api_version": "core/v2",
  "spec": {
    "opt_out": false
  }
}
```

Tessen metrics log examples

For unlicensed instances that opt in to Tessen and all licensed instances, Sensu sends various metrics back to the Tessen service. In the example metrics log below, Sensu is sending the number of check hooks back to the Tessen service.

```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:

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "handler_count",
    "metric_value": 10,
    "msg": "collected a metric for tessen",
    "time": "2019-09-16T09:02:06Z"
}
```

Or the number of filters:

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "filter_count",
    "metric_value": 4,
    "msg": "collected a metric for tessen",
    "time": "2019-09-16T09:02:01Z"
}
```

Or the number of authentication providers, secrets providers, and secrets:

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "auth_provider_count",
    "metric_value": 2,
    "msg": "collected a metric for tessen",
    "time": "2020-03-30T15:16:42-04:00"
}
```

```
{
    "component": "tessend",
    "level": "debug",
    "metric_name": "secret_provider_count",
    "metric_value": 1,
    "msg": "collected a metric for tessen",
```

```
        "time": "2020-03-30T15:17:12-04:00"
}


{
    "component": "tessend",
    "level": "debug",
    "metric_name": "secret_count",
    "metric_value": 1,
    "msg": "collected a metric for tessen",
    "time": "2020-03-30T15:16:17-04:00"
}
```

If you opt into Tessen, you can view all of the metrics in the logs:

```
journalctl _COMM=sensu-backend.service
```

To view the events on-disk, read Log Sensu services with systemd.

Manage Secrets

Sensu's secrets management eliminates the need to expose secrets like usernames, passwords, and access keys in your Sensu configuration. Secrets management is available for Sensu handler, mutator, and check resources.

Use secrets management in Sensu explains how to use Sensu's built-in secrets provider (`Env`) or HashiCorp Vault as your external secrets provider and authenticate without exposing your secrets. Follow this guide to set up your PagerDuty Integration Key as a secret and create a PagerDuty handler definition that requires the secret. Your Sensu backend will be able to execute the handler with any check.

Secrets

Secrets are configured via secrets resources. A secret resource definition refers to the secrets provider (`Env` or `VaultProvider`) and an ID (the named secret to fetch from the secrets provider).

The secrets reference includes the specification, sensuctl configuration subcommands, and examples for secrets resources.

Secrets providers

The Sensu Go commercial distribution includes a built-in secrets provider, `Env`, that exposes secrets from environment variables on your Sensu backend nodes. You can also use the secrets provider `VaultProvider` to authenticate via the HashiCorp Vault integration.

The secrets providers reference includes the resource specification, instructions for retrieving your secrets providers configuration via the Sensu API, and examples.

Use secrets management in Sensu

> **COMMERCIAL FEATURE**: Access the Env and VaultProvider secrets provider datatypes in the packaged Sensu Go distribution. For more information, read Get started with commercial features.

Sensu's secrets management allows you to avoid exposing secrets like usernames, passwords, and access keys in your Sensu configuration. In this guide, you'll learn how to use Sensu's built-in secrets provider, `Env`, or HashiCorp Vault as your external secrets provider and authenticate without exposing your secrets. You'll set up your PagerDuty Integration Key as a secret and create a PagerDuty handler definition that requires the secret. Your Sensu backend can then execute the handler with any check.

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

Secrets are configured via secrets resources. A secret resource definition refers to the secrets provider (`Env` or `VaultProvider`) and an ID (the named secret to fetch from the secrets provider).

This guide only covers the handler use case, but you can use secrets management in handler, mutator, and check execution. When a check configuration references a secret, the Sensu backend will only transmit the check's execution requests to agents that are connected via mutually authenticated transport layer security (mTLS)-encrypted WebSockets. Read more about enabling mTLS.

The secret included in your Sensu handler will be exposed to Sensu services at runtime as an environment variable. Sensu only exposes secrets to Sensu services like environment variables and automatically redacts secrets from all logs, the API, and the web UI.
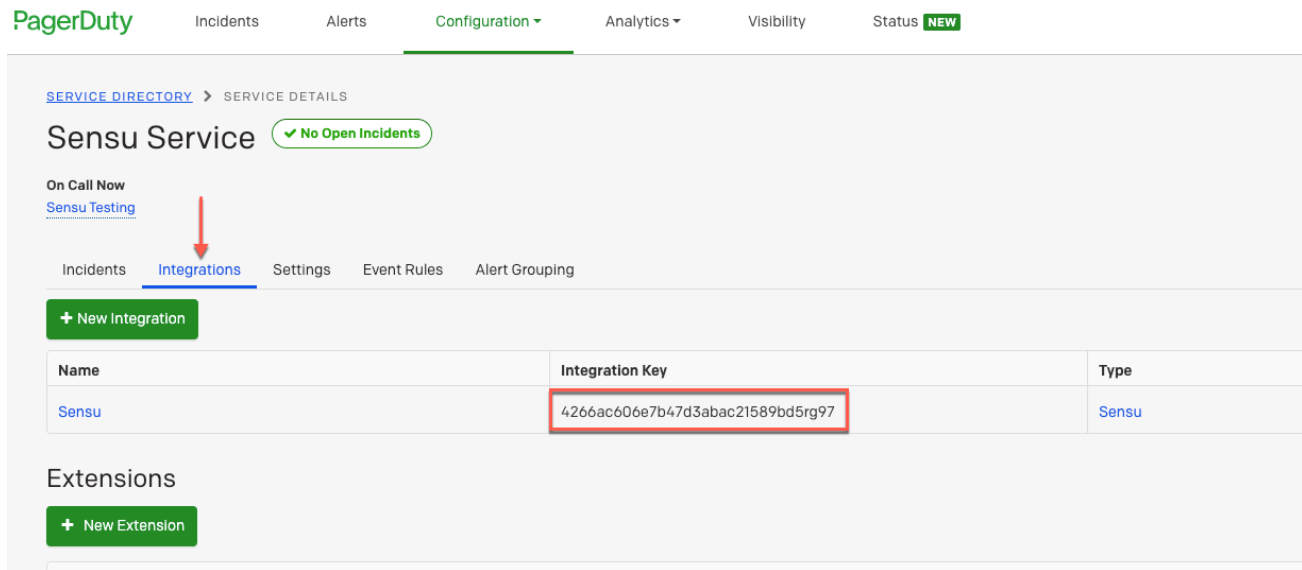
Retrieve your PagerDuty Integration Key

The example in this guide uses the PagerDuty Integration Key as a secret and a PagerDuty handler definition that requires the secret.

Here's how to find your Integration Key in PagerDuty so you can set it up as your secret:

1. Log in to your PagerDuty account.
2. In the **Configuration** drop-down menu, select **Services**.

3. Click your Sensu service.
4. Click the **Integrations** tab. The Integration Key is listed in the second column.

Make a note of your Integration Key — you'll need it to create your <u>backend environment variable</u> or <u>HashiCorp Vault secret</u>.

Use Env for secrets management

The <u>Sensu Go commercial distribution</u> includes a built-in secrets provider, `Env`, that exposes secrets from <u>environment variables</u> on your Sensu backend nodes. The `Env` secrets provider is automatically created with an empty `spec` when you start your Sensu backend.

Create your backend environment variable

To use the built-in `Env` secrets provider, you will add your secret as a backend environment variable.

First, make sure you have created the files you need to store <u>backend environment variables</u>.

Then, run the following code, replacing `INTEGRATION_KEY` with your PagerDuty Integration Key:

SHELL

```
echo 'SENSU_PAGERDUTY_KEY=INTEGRATION_KEY' | sudo tee -a /etc/default/sensu-backend
```

SHELL

```
echo 'SENSU_PAGERDUTY_KEY=INTEGRATION_KEY' | sudo tee -a /etc/sysconfig/sensu-backend
```

Restart the sensu-backend:

```
sudo systemctl restart sensu-backend
```

This configures the `SENSU_PAGERDUTY_KEY` environment variable to your PagerDuty Integration Key in the context of the sensu-backend process.

Create your Env secret

Now you'll use `sensuctl create` to create your secret. This code creates a secret named `pagerduty_key` that refers to the environment variable ID `SENSU_PAGERDUTY_KEY`. Run:

SHELL

```
cat << EOF | sensuctl create
---
type: Secret
api_version: secrets/v1
metadata:
  name: pagerduty_key
  namespace: default
spec:
  id: SENSU_PAGERDUTY_KEY
  provider: env
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "pagerduty_key",
    "namespace": "default"
```

```
    },
    "spec": {
      "id": "SENSU_PAGERDUTY_KEY",
      "provider": "env"
    }
  }
  EOF
```

You can securely pass your PagerDuty Integration Key in Sensu checks, handlers, and mutators by referring to the `pagerduty_key` secret.Skip to the add a handler section, where you'll use your `pagerduty_key` secret in your handler definition.

Use HashiCorp Vault for secrets management

This section explains how to use HashiCorp Vault as your external secrets provider to authenticate via the HashiCorp Vault integration's token auth method or TLS certificate auth method.

> **NOTE**: You must set up HashiCorp Vault to use `VaultProvider` secrets management in production.The examples in this guide use the Vault dev server, which is useful for learning and experimenting.The Vault dev server gives you access to a preconfigured, running Vault server with in-memory storage that you can use right away.Follow the HashiCorp Learn curriculum when you are ready to set up a production server in Vault.
>
> In addition, this guide uses the Vault KV secrets engine.Using the Vault KV secrets engine with the Vault dev server requires v2 connections.For this reason, in the `VaultProvider` spec in these examples, the client `version` value is **v2**.

Configure your Vault authentication method (token or TLS)

If you use HashiCorp Vault as your external secrets provider, you can authenticate via the HashiCorp Vault integration's token or transport layer security (TLS) certificate authentication method.

Vault token authentication

Follow the steps in this section to use HashiCorp Vault as your external secrets provider to authenticate with the HashiCorp Vault integration's token auth method.

Retrieve your Vault root token

To retrieve your Vault root token:

1. _Download and install_ the Vault edition for your operating system.
2. Open a terminal window and run `vault server -dev`.

The command output includes a `Root Token` line. Find this line in your command output and copy the `Root Token` value. You will use it next to create your Vault secrets provider.

```
WARNING! dev mode is enabled! In this mode, Vault runs entirely in-memory
and starts unsealed with a single unseal key. The root token is already
authenticated to the CLI, so you can immediately begin using Vault.

You may need to set the following environment variable:

    $ export VAULT_ADDR='http://127.0.0.1:8200'

The unseal key and root token are displayed below in case you want to
seal/unseal the Vault or re-authenticate.

Unseal Key: hBYzDEg7M/uVawGn6unNG4boUSZL+kf8odYZhCtGe4Q=
Root Token: s.tG3vkbOvcBLC63Bn7Mb729oF

Development mode should NOT be used in production installations!

==> Vault server started! Log data will stream in below:
```

Leave the Vault dev server running. Because you aren't using TLS, you will need to set `VAULT_ADDR=http://127.0.0.1:8200` in your shell environment.

Create your Vault secrets provider

Use `sensuctl create` to create your secrets provider, `vault`. In the code below, replace `<root_token>` with the `Root Token` value for your Vault dev server. Then, run:

SHELL

```shell
cat << EOF | sensuctl create
---
type: VaultProvider
api_version: secrets/v1
metadata:
  name: vault
spec:
  client:
    address: http://localhost:8200
    token: <root_token>
    version: v2
    tls: null
    max_retries: 2
    timeout: 20s
    rate_limiter:
      limit: 10
      burst: 100
EOF
```

SHELL

```shell
cat << EOF | sensuctl create
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "vault"
  },
  "spec": {
    "client": {
      "address": "http://localhost:8200",
      "token": "<root_token>",
      "version": "v2",
      "tls": null,
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10,
        "burst": 100
      }
    }
```

```
      }
   }
   EOF
```

To continue, skip ahead to create your Vault secret.

Vault TLS certificate authentication

This section explains how use HashiCorp Vault as your external secrets provider to authenticate with the HashiCorp Vault integration's TLS certificate auth method.

> **NOTE**: You will need to set up *HashiCorp Vault* in production to use TLS certificate-based authentication. In Vault's dev server, TLS is not enabled. Follow the *HashiCorp Learn curriculum* when you are ready to set up a production server in Vault.

First, in your Vault, enable and configure certificate authentication. For example, your Vault might be configured for certificate authentication like this:

```
vault write auth/cert/certs/sensu-backend \
    display_name=sensu-backend \
    policies=sensu-backend-policy \
    certificate=@sensu-backend-vault.pem \
    ttl=3600
```

Second, configure your `VaultProvider` in Sensu:

YML

```
---
type: VaultProvider
api_version: secrets/v1
metadata:
  name: vault
spec:
  client:
    address: https://vault.example.com:8200
    version: v2
```

```
    tls:
      ca_cert: /path/to/your/ca.pem
      client_cert: /etc/sensu/ssl/sensu-backend-vault.pem
      client_key: /etc/sensu/ssl/sensu-backend-vault-key.pem
      cname: sensu-backend.example.com
    max_retries: 2
    timeout: 20s
    rate_limiter:
      limit: 10
      burst: 100
```

SHELL

```
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "vault"
  },
  "spec": {
    "client": {
      "address": "https://vault.example.com:8200",
      "version": "v2",
      "tls": {
        "ca_cert": "/path/to/your/ca.pem",
        "client_cert": "/etc/sensu/ssl/sensu-backend-vault.pem",
        "client_key": "/etc/sensu/ssl/sensu-backend-vault-key.pem",
        "cname": "sensu-backend.example.com"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10,
        "burst": 100
      }
    }
  }
}
```

The certificate you specify for `tls.client_cert` should be the same certificate you configured in

your Vault for certificate authentication.

Next, create your Vault secret.

Create your Vault secret

First, retrieve your PagerDuty Integration Key (the secret you will set up in Vault).

Next, open a new terminal and run `vault kv put secret/pagerduty key=<integration_key>`.
Replace `<integration_key>` with your PagerDuty Integration Key. This writes your secret into Vault.

In this example, the name of the secret is `pagerduty`. The `pagerduty` secret contains a key, and you
specified that the `key` value is your PagerDuty Integration Key.

> **NOTE**: The Vault dev server is preconfigured with the secret keyspace already set up, so we
> recommend using the `secret/` path for the `id` value while you are learning and getting started
> with Vault secrets management.
>
> This example uses the `id` format for the Vault KV Secrets Engine v1: `secret/pagerduty#key`. If
> you are using the Vault KV Secrets Engine v2, the format is `secrets/sensu#pagerduty#key`.

Run `vault kv get secret/pagerduty` to view the secret you just set up.

Use `sensuctl create` to create your `vault` secret:

SHELL

```
cat << EOF | sensuctl create
---
type: Secret
api_version: secrets/v1
metadata:
  name: pagerduty_key
  namespace: default
spec:
  id: secret/pagerduty#key
  provider: vault
EOF
```

SHELL

```
cat << EOF | sensuctl create
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "pagerduty_key",
    "namespace": "default"
  },
  "spec": {
    "id": "secret/pagerduty#key",
    "provider": "vault"
  }
}
EOF
```

Now you can securely pass your PagerDuty Integration Key in the handlers, and mutators by referring to the `pagerduty_key` secret.In the add a handler section, you'll use your `pagerduty_key` secret in your handler definition.

Add a handler

Register the PagerDuty Handler dynamic runtime asset

To begin, register the Sensu PagerDuty Handler dynamic runtime asset with `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 dynamic runtime asset: `pagerduty-handler` .

NOTE: You can adjust the dynamic runtime asset definition according to your Sensu configuration if needed.

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

With this handler, Sensu can trigger and resolve PagerDuty incidents.However, you still need to add your secret to the handler spec so that it requires your backend to request secrets from your secrets provider.

Add your secret to the handler spec

To create a handler definition that uses your `pagerduty_key` secret, run:

SHELL

```
cat << EOF | sensuctl create
---
api_version: core/v2
type: Handler
metadata:
  namespace: default
  name: pagerduty
spec:
  type: pipe
  command: pagerduty-handler --token $PD_TOKEN
  secrets:
  - name: PD_TOKEN
    secret: pagerduty_key
  runtime_assets:
  - pagerduty-handler
  timeout: 10
  filters:
  - is_incident
EOF
```

SHELL

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

```
      "name": "pagerduty"
    },
    "spec": {
      "type": "pipe",
      "command": "pagerduty-handler --token $PD_TOKEN",
      "secrets": [
        {
          "name": "PD_TOKEN",
          "secret": "pagerduty_key"
        }
      ],
      "runtime_assets": [
        "pagerduty-handler"
      ],
      "timeout": 10,
      "filters": [
        "is_incident"
      ]
    }
  }
  EOF
```

Now that your handler is set up and Sensu can create incidents in PagerDuty, you can automate this workflow by adding your `pagerduty` handler to your Sensu service check definitions.Read Monitor server resources to learn more.

Next steps

Read the secrets or secrets providers reference for in-depth secrets management documentation.

Secrets reference

Sensu's secrets management eliminates the need to expose secrets in your Sensu configuration. When a Sensu resource definition requires a secret (for example, a username or password), Sensu allows you to obtain secrets from one or more external secrets providers, so you can both refer to external secrets and consume secrets via backend environment variables.

> **NOTE**: Secrets management is implemented for *checks*, *handlers*, and *mutators*.

Only Sensu backends have access to request secrets from a secrets provider. Sensu backends cache fetched secrets in memory, with no persistence to a Sensu datastore or file on disk. Secrets provided via a "lease" with a "lease duration" are deleted from Sensu's in-memory cache after the configured number of seconds, prompting the Sensu backend to request the secret again.

Secrets are only transmitted over a transport layer security (TLS) WebSocket connection. Unencrypted connections must not transmit privileged information. For checks, hooks, and dynamic runtime assets, you must enable mutual TLS (mTLS). Sensu will not transmit secrets to agents that do not use mTLS.

Sensu only exposes secrets to Sensu services like environment variables and automatically redacts secrets from all logs, the API, and the web UI.

Secret examples

A secret resource definition refers to a secrets `id` and a secrets `provider`. Read the secrets provider reference for the provider specification.

YML

```
---
type: Secret
api_version: secrets/v1
metadata:
```

```
    name: sensu-ansible-token
    namespace: default
  spec:
    id: ANSIBLE_TOKEN
    provider: env
```

JSON

```json
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default"
  },
  "spec": {
    "id": "ANSIBLE_TOKEN",
    "provider": "env"
  }
}
```

Configure secrets that target a HashiCorp Vault as shown in the following example:

YML

```yml
---
type: Secret
api_version: secrets/v1
metadata:
  name: sensu-ansible
  namespace: default
spec:
  id: 'secret/database#password'
  provider: vault
```

JSON

```json
{
  "type": "Secret",
  "api_version": "secrets/v1",
```

```
    "metadata": {
      "name": "sensu-ansible",
      "namespace": "default"
    },
    "spec": {
      "id": "secret/database#password",
      "provider": "vault"
    }
  }
```

The `id` value for secrets that target a HashiCorp Vault must start with the name of the secret's path in Vault. The Vault dev server is preconfigured with the secret keyspace already set up. This is convenient for learning and getting started with Vault secrets management, so this example and our guide to secrets management use the `secret/` path for the `id` value. In this example, the name of the secret is `database`. The database secret contains a key called `password,` and its value is the password to our database.

Secret configuration

You can use the Secrets API and sensuctl to create, view, and manage your secrets configuration. To manage secrets configuration with sensuctl, configure sensuctl as the default `admin` user.

The standard sensuctl subcommands are available for secrets (list, info, and delete).

To list all secrets:

```
sensuctl secret list
```

To review a secret's status:

```
sensuctl secret info SECRET_NAME
```

To delete a secret:

```
sensuctl secret delete SECRET_NAME
```

`SECRET_NAME` is the value specified in the secret's `name` metadata attribute.

Secret specification

Top-level attributes

type	
description	Top-level attribute that specifies the resource type. For secrets configuration, the type should always be `Secret`.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String

example	**YML**
	
	```type: Secret```
	
	JSON
	
	```{ "type": "Secret" }```

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For secrets configuration in this version of Sensu, the api_version should always be `secrets/v1`.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.

type	String

YML

```
api_version: secrets/v1
```

JSON

```
{
  "api_version": "secrets/v1"
}
```

metadata

description	Top-level scope that contains the secret's `name` and `namespace` as well as the `created_by` field.
required	true
type	Map of key-value pairs

YML

```
metadata:
  name: sensu-ansible-token
  namespace: default
  created_by: admin
```

JSON

```
{
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

spec	
description	Top-level map that includes secrets configuration spec attributes.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs

YML

```yaml
spec:
  id: ANSIBLE_TOKEN
  provider: env
```

JSON

```json
{
  "spec": {
    "id": "ANSIBLE_TOKEN",
    "provider": "env"
  }
}
```

Metadata attributes

name	
description	Name for the secret that is used internally by Sensu.
required	true
type	String

YML

```yaml
name: sensu-ansible-token
```

JSON

```
{
  "name": "sensu-ansible-token"
}
```

namespace

description	Sensu RBAC namespace that the secret belongs to.
required	true
type	String

example

YML

```
namespace: default
```

JSON

```
{
  "namespace": "default"
}
```

created_by

description	Username of the Sensu user who created the secret or last updated the secret. Sensu automatically populates the `created_by` field when the secret is created or updated.
required	false
type	String

example

YML

```
created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

Spec attributes

id	
description	Identifying key for the provider to retrieve the secret. For the `Env` secrets provider, the `id` is the environment variable. For the `Vault` secrets provider, the `id` specifies the secrets engine path, the path to the secret within that secrets engine, and the field to retrieve within the secret.
required	true
type	String

example for Vault KV Secrets Engine v1	**YML**
	
	`id: secret/ansible#token`
	
	JSON
	
	`{`
	`  "id": "secret/ansible#token"`
	`}`

example for Vault KV Secrets Engine v2	**YML** `id: secrets/sensu#ansible#token` **JSON**

```
{
  "id": "secrets/sensu#ansible#token"
}
```

provider

description	Name of the provider with the secret.
required	true
type	String

YML

example

```
provider: vault
```

JSON

```
{
  "provider": "vault"
}
```

Secrets providers reference

Sensu's secrets management eliminates the need to expose secrets like usernames, passwords, and access keys in your Sensu configuration. With Sensu's secrets management, you can obtain secrets from one or more external secrets providers, refer to external secrets, and consume secrets via [backend environment variables](#).

NOTE: *Secrets management is implemented for [checks](#), [handlers](#), and [mutators](#).*

Only Sensu backends have access to request [secrets](#) from a secrets provider. Secrets are only transmitted over a transport layer security (TLS) WebSocket connection. Unencrypted connections must not transmit privileged information. For checks, hooks, and dynamic runtime assets, you must [enable mutual TLS (mTLS)](#). Sensu will not transmit secrets to agents that do not use mTLS.

The [Sensu Go commercial distribution](#) includes a built-in secrets provider, `Env`, that exposes secrets from [environment variables](#) on your Sensu backend nodes. You can also use the secrets provider `VaultProvider` to authenticate via the HashiCorp Vault integration's [token auth method](#) or [TLS certificate auth method](#).

You can configure any number of `VaultProvider` secrets providers. However, you can only have a single `Env` secrets provider: the one that is included with the Sensu Go [commercial distribution](#).

Secrets providers are cluster-wide resources and compatible with generic functions.

VaultProvider secrets provider example

The `VaultProvider` secrets provider is a vendor-specific implementation for [HashiCorp Vault](#) secrets management.

YML

```
---
```

```yaml
type: VaultProvider
api_version: secrets/v1
metadata:
  name: vault
spec:
  client:
    address: https://vaultserver.example.com:8200
    token: VAULT_TOKEN
    version: v1
    tls:
      ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
    max_retries: 2
    timeout: 20s
    rate_limiter:
      limit: 10
      burst: 100
```

JSON

```json
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "vault"
  },
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
```

Env secrets provider example

Sensu's built-in `Env` secrets provider exposes secrets from <u>backend environment variables</u>. The `Env` secrets provider is automatically created with an empty `spec` when you start your Sensu backend.

Using the `Env` secrets provider may require you to synchronize environment variables in Sensu backend clusters. The <u>Use secrets management</u> guide demonstrates how to configure the `Env` secrets provider.

YML

```
---
type: Env
api_version: secrets/v1
metadata:
  name: env
spec: {}
```

JSON

```
{
  "type": "Env",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "env"
  },
  "spec": {}
}
```

Secrets providers configuration

You can use the <u>Secrets API</u> to create, view, and manage your secrets providers configuration.

For example, to retrieve the list of secrets providers:

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers \
-H "Authorization: Key $SENSU_API_KEY"
```

Secrets providers specification

> **NOTE**: The attribute descriptions in this section use the `VaultProvider` datatype. The _secrets providers examples_ section includes an example for the `Env` datatype.

Top-level attributes

type	
description	Top-level attribute that specifies the resource type. May be either `Env` (if you are using Sensu's built-in secrets provider) or `VaultProvider` (if you are using HashiCorp Vault as the secrets provider).
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String

example	**YML**
	```type: VaultProvider```
	JSON
	```{ "type": "VaultProvider" }```

api_version	
description	Top-level attribute that specifies the Sensu API group and version. For

	secrets configuration in this version of Sensu, the api_version should always be `secrets/v1`.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	String

YML

example

```yml
api_version: secrets/v1
```

JSON

```json
{
  "api_version": "secrets/v1"
}
```

metadata

description	Top-level scope that contains the secrets provider `name` and `created_by` field. Namespace is not supported in the metadata because secrets providers are cluster-wide resources.
required	true
type	Map of key-value pairs

YML

example

```yml
metadata:
  name: vault
  created_by: admin
```

JSON

```json
{
  "metadata": {
    "name": "vault",
    "created_by": "admin"
  }
}
```

```
      }
```

spec

description	Top-level map that includes secrets provider configuration spec attributes.
required	Required for secrets configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs

YML

example	

```yaml
spec:
  client:
    address: https://vaultserver.example.com:8200
    max_retries: 2
    rate_limiter:
      limit: 10
      burst: 100
    timeout: 20s
    tls:
      ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
    token: VAULT_TOKEN
    version: v1
```

JSON

```json
{
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "max_retries": 2,
      "rate_limiter": {
        "limit": 10,
        "burst": 100
      },
      "timeout": "20s",
      "tls": {
```

```
          "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
        },
        "token": "VAULT_TOKEN",
        "version": "v1"
      }
    }
  }
```

Metadata attributes

name	
description	Provider name used internally by Sensu.
required	true
type	String

YML

```yml
name: vault
```

JSON

```json
{
  "name": "vault"
}
```

created_by	
description	Username of the Sensu user who created the secrets provider or last updated the secrets provider. Sensu automatically populates the `created_by` field when the secrets provider is created or updated.
required	false

type	String

YML

example

```yaml
created_by: admin
```

JSON

```json
{
  "created_by": "admin"
}
```

Spec attributes

client	
description	Map that includes secrets provider configuration <u>client attributes</u>.
required	true
type	Map of key-value pairs

YML

example

```yaml
client:
  address: https://vaultserver.example.com:8200
  max_retries: 2
  rate_limiter:
    limit: 10
    burst: 100
  timeout: 20s
  tls:
    ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
  token: VAULT_TOKEN
  version: v1
```

JSON

```json
{
  "client": {
    "address": "https://vaultserver.example.com:8200",
    "max_retries": 2,
    "rate_limiter": {
      "limit": 10,
      "burst": 100
    },
    "timeout": "20s",
    "tls": {
      "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
    },
    "token": "VAULT_TOKEN",
    "version": "v1"
  }
}
```

Client attributes

address	
description	Vault server address.
required	true
type	String

example	**YML**
	
	``yml
	address: https://vaultserver.example.com:8200
	``
	
	JSON
	
	``json
	{
	  "address": "https://vaultserver.example.com:8200"
	}
	``

max_retries

description	Number of times to retry connecting to the vault provider.
required	true
type	Integer
default	2

example

YML

```yaml
max_retries: 2
```

JSON

```json
{
  "max_retries": 2
}
```

rate_limiter

description	Maximum rate and burst limits for the secrets API.
required	false
type	Map of key-value pairs

example

YML

```yaml
rate_limiter:
  limit: 10
  burst: 100
```

JSON

```json
{
  "rate_limiter": {
```

```
      "limit": 10,
      "burst": 100
    }
  }
```

timeout

description	Provider connection timeout (hard stop).
required	false
type	String
default	60s

example	**YML**

```
  timeout: 20s
```

JSON

```
  {
    "timeout": "20s"
  }
```

tls

description	TLS object. Vault only works with TLS configured. You may need to set up a CA cert if it is not already stored in your operating system's trust store. To do this, set the TLS object and provide the `ca_cert` path. You may also need to set up `client_cert`, `client_key`, or `cname`.
required	false
type	Map of key-value pairs

YML

example	

```yaml
tls:
  ca_cert: "/etc/ssl/certs/vault_ca_cert.pem"
  client_cert: "/etc/ssl/certs/vault_cert.pem"
  client_key: "/etc/ssl/certs/vault_key.pem"
  cname: vault_client.example.com
```

JSON

```json
{
  "tls": {
    "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem",
    "client_cert": "/etc/ssl/certs/vault_cert.pem",
    "client_key": "/etc/ssl/certs/vault_key.pem",
    "cname": "vault_client.example.com"
  }
}
```

token

description	Vault token to use for authentication.
required	true
type	String

YML

```yaml
token: VAULT_TOKEN
```

JSON

```json
{
  "token": "VAULT_TOKEN"
}
```

version	
description	HashiCorp Vault key/value store version.
required	true
type	String
allowed values	`v1` and `v2`
example	**YML**

```
version: v1
```

JSON

```
{
    "version": "v1"
}
``` |

Rate limiter attributes

limit	
description	Maximum number of secrets requests per second that can be transmitted to the backend with the secrets API.
required	false
type	Float
example	**YML**

```
limit: 10.0
```

JSON

```
{
    "limit": 10.0
``` |

```
    }
```

burst

description	Maximum amount of burst allowed in a rate interval for the secrets API.
required	false
type	Integer
example	**YML**

```yml
burst: 100
```

JSON

```json
{
  "burst": 100
}
```

Guides Index

This index links to every guide in the Sensu documentation.Guides describe how to configure Sensu to complete specific observability tasks and workflows to monitor server resources, route alerts and reduce alert fatigue, export metrics, plan maintenance windows, and more, with examples and step-by-step walkthroughs.

- Aggregate metrics with the Sensu StatsD listener

- Augment event data with check hooks

- Collect Prometheus metrics with Sensu

- Collect service metrics with Sensu checks

- Create a read-only user with role-based access control (RBAC)

- Create handler templates

- Create limited service accounts with role-based access control (RBAC)

- Generate certificates for your Sensu installation

- Log Sensu services with systemd

- Monitor external resources with proxy entities

- Monitor Sensu with Sensu

- Monitor server resources with checks

- Multi-cluster visibility with federation

- Plan maintenance windows with silencing

- Populate metrics in InfluxDB with handlers

- Reduce alert fatigue with event filters

- Route alerts with event filters

- Run a Sensu cluster

- Scale Sensu Go with Enterprise datastore

- Secure Sensu

- Send email alerts with the Sensu Go Email Handler

-

Sensuctl CLI

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. For Windows operating systems, sensuctl uses `cmd.exe` for the execution environment. For all other operating systems, sensuctl uses the Bourne shell (sh).

Read Install Sensu to install and configure sensuctl.

First-time setup and authentication

To 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
```

Sensuctl uses your username and password to obtain access and refresh tokens via the Sensu authentication API. The access and refresh tokens are HMAC-SHA256 JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature.

Upon successful authentication, sensuctl stores the access and refresh tokens in a "cluster" configuration file under the current user's home directory. For example, on Unix systems, sensuctl stores the tokens in `$HOME/.config/sensu/sensuctl/cluster`.

Sensu backend URL

The Sensu backend URL is the HTTP or HTTPS URL where sensuctl can connect to the Sensu backend server. The default URL is `http://127.0.0.1:8080`.

To connect to a Sensu cluster, connect sensuctl to any single backend in the cluster. For information about configuring the Sensu backend URL, read the backend reference.

Configuration files

During configuration, sensuctl creates configuration files that contain information for connecting to your Sensu Go deployment. You can find these files at `$HOME/.config/sensu/sensuctl/profile` and `$HOME/.config/sensu/sensuctl/cluster`.

Use the `cat` command to view the contents of these files. For example, to view your sensuctl profile configuration, run:

```
cat .config/sensu/sensuctl/profile
```

The response should be similar to this example:

```
{
  "format": "tabular",
  "namespace": "default",
  "username": "admin"
}
```

To view your sensuctl cluster configuration, run:

```
cat .config/sensu/sensuctl/cluster
```

The response should be similar to this example:

```
{
  "api-url": "http://localhost:8080",
  "trusted-ca-file": "",
  "insecure-skip-tls-verify": false,
  "access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "expires_at": 1550082282,
  "refresh_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

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.

Username, password, and namespace

During the Sensu backend installation process, you create an administrator username and password and 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, read the RBAC reference.

> **NOTE**: For a **new** installation, you can set administrator credentials with environment variables during _initialization_. If you are using Docker and you do not include the environment variables to set administrator credentials, the backend will initialize with the default username (`admin`) and password (`P@ssw0rd!`).

Your ability to get, list, create, update, and delete resources with sensuctl depends on the permissions assigned to your Sensu user. For more information about configuring Sensu access control, read the RBAC reference.

Change admin user's password

After you have configured sensuctl and authenticated, you can change the admin user's password. Run

```
sensuctl user change-password --interactive
```

You must specify the user's current password to use the `sensuctl user change-password` command.

Reset a user password

To reset a user password without specifying the current password, run:

```
sensuctl user reset-password USERNAME --interactive
```

You must have admin permissions to use the `sensuctl user reset-password` command.

Test a user password

To test the password for a user created with Sensu's built-in basic authentication:

```
sensuctl user test-creds USERNAME --password 'password'
```

An empty response indicates valid credentials. A `request-unauthorized` response indicates invalid credentials.

> **NOTE**: The `sensuctl user test-creds` command tests passwords for users created with Sensu's built-in basic authentication provider. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).

For example, if you test LDAP credentials with the `sensuctl user test-creds` command, the backend will log an error, even if 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"}
```

Generate a password hash

To generate a password hash for a specified cleartext password, run:

```
sensuctl user hash-password PASSWORD
```

The `sensuctl user hash-password` command creates a bcrypt hash of the specified password. You can use this hash instead of the password when you use sensuctl to create and edit users.

Preferred output format

Preferred output format

Sensuctl supports the following output formats:

- `tabular` : Output is organized in user-friendly columns (default).

- `yaml` : Output is in YAML format. Resource definitions include an outer-level `spec` "wrapping" for resource attributes and list the resource `type` and `api_version` .

- `wrapped-json` : Output is in JSON format. Resource definitions include an outer-level `spec` "wrapping" for resource attributes and list the resource `type` and `api_version` .

- `json` : Output is in JSON format. Resource definitions **do not** include an outer-level `spec` "wrapping" or the resource `type` and `api_version` .

After you are logged in, you can change the default output format with `sensuctl config set-format` or set the output format per command with the `--format` flag.

Output format significance

To use sensuctl create to create a resource, you must provide the resource definition in `yaml` or `wrapped-json` format. These formats include the resource type, which sensuctl needs to determine what kind of resource to create.

The Sensu API uses `json` output format for responses for APIs in the `core` group. For APIs that are not in the `core` group, responses are in the `wrapped-json` output format.

Sensu sends events to the backend in `json` [format](#), without the `spec` attribute wrapper or `type` and `api_version` attributes.

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
```

Get help

Sensuctl supports a `--help` flag for each command and subcommand.

List command and global flags

```
sensuctl --help
```

List subcommands and flags

```
sensuctl check --help
```

List 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, read the RBAC reference.

For example, to change the default namespace to `development`:

```
sensuctl config set-namespace development
```

Log out of sensuctl

To log out of sensuctl:

```
sensuctl logout
```

To log back in to sensuctl:

```
sensuctl configure
```

View the sensuctl version number

To display the current version of sensuctl:

```
sensuctl version
```

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 (default
"default")
  --timeout duration          timeout when communicating with sensu backend (default
```

```
15s)
--trusted-ca-file string     TLS CA certificate bundle in PEM format
```

You can permanently set these flags by editing `.config/sensu/sensuctl/{cluster, profile}`.

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
```

Create and manage resources with sensuctl

Use the sensuctl command line tool to create and manage resources within Sensu. Sensuctl works by calling Sensu's underlying API to create, read, update, and delete resources, events, and entities.

Create resources

The `sensuctl create` command allows you to create or update resources by reading from STDIN or a flag configured file (`-f`). The `create` command accepts Sensu resource definitions in `yaml` or `wrapped-json` formats, which wrap the contents of the resource in `spec` and identify the resource `type` and `api_version`. Review the list of supported resource types `for sensuctl create`. Read the reference docs for information about creating resource definitions.

These examples specify two resources: a `marketing-site` check and a `slack` handler. In the YAML example, the resources are separated by a line with three hyphens: `---`. In the wrapped JSON example, the resources are separated *without* a comma.

Save these resource definitions to a file named `my-resources.yml` or `my-resources.json` :

YML

```
---
type: CheckConfig
api_version: core/v2
metadata:
  name: marketing-site
  namespace: default
spec:
  command: http-check -u https://sensu.io
  subscriptions:
  - demo
  interval: 15
  handlers:
  - slack
---
```

```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/XXXXXXXXXXXXXXXXXXXXXXXXX
  filters:
  - is_incident
  - not_silenced
  type: pipe
```

SHELL

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata" : {
    "name": "marketing-site",
    "namespace": "default"
    },
  "spec": {
    "command": "http-check -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 these resources from `my-resources.yml` or `my-resources.json` with `sensuctl create` :

SHELL

```
sensuctl create --file my-resources.yml
```

SHELL

```
sensuctl create --file my-resources.json
```

Or:

SHELL

```
cat my-resources.yml | sensuctl create
```

SHELL

```
cat my-resources.json | sensuctl create
```

sensuctl create flags

Run `sensuctl create -h` to view command-specific and global flags. The following table describes the command-specific flags.

Flag	Function and important notes
`-f` or `--file`	Files, URLs, or directories to create resources from. Strings.
`-h` or `--help`	Help for the create command.
`-r` or `--recursive`	Create command will follow subdirectories.

sensuctl create resource types

sensuctl create types			
`AdhocRequest`	`adhoc_request`	`Asset`	`asset`
`CheckConfig`	`check_config`	`ClusterRole`	`cluster_role`
`ClusterRoleBinding`	`cluster_role_binding`	`Entity`	`Env`
`entity`	`EtcdReplicators`	`Event`	`event`
`EventFilter`	`event_filter`	`GlobalConfig`	`Handler`
`handler`	`Hook`	`hook`	`HookConfig`
`hook_config`	`Mutator`	`mutator`	`Namespace`
`namespace`	`Role`	`role`	`RoleBinding`
`role_binding`	`Secret`	`Silenced`	`silenced`
`User`	`user`	`VaultProvider`	`ldap`
`ad`	`oidc`	`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, read the namespaces reference. The RBAC reference includes a list of namespaced resource types.

The `sensuctl create` command applies namespaces to resources in the following order, from highest precedence to lowest:

1. **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. Read Manage sensuctl to view your current session config and set the session namespace.

This example defines a handler *without* a `namespace` attribute:

YML

```yaml
---
type: Handler
api_version: core/v2
metadata:
  name: pagerduty
spec:
  command: sensu-pagerduty-handler
  env_vars:
  - PAGERDUTY_TOKEN=SECRET
  type: pipe
```

JSON

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

```
  "metadata": {
    "name": "pagerduty"
  },
  "spec": {
    "command": "sensu-pagerduty-handler",
    "env_vars": [
      "PAGERDUTY_TOKEN=SECRET"
    ],
    "type": "pipe"
  }
}
```

If this resource definition is saved in a file named `pagerduty.yml` or `pagerduty.json`, create the `pagerduty` handler in the `default` namespace with this command:

SHELL

```
sensuctl create --file pagerduty.yml --namespace default
```

SHELL

```
sensuctl create --file pagerduty.json --namespace default
```

To create the `pagerduty` handler in the `production` namespace:

SHELL

```
sensuctl create --file pagerduty.yml --namespace production
```

SHELL

```
sensuctl create --file pagerduty.json --namespace production
```

To create the `pagerduty` handler in the current session namespace:

SHELL

```
sensuctl create --file pagerduty.yml
```

SHELL

```
sensuctl create --file pagerduty.json
```

Delete resources

The `sensuctl delete` command allows you to delete resources by reading from STDIN or a 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` or `my-resources.json` with `sensuctl delete` :

SHELL

```
sensuctl delete --file my-resources.yml
```

SHELL

```
sensuctl delete --file my-resources.json
```

Or:

SHELL

```
cat my-resources.yml | sensuctl delete
```

SHELL

```
cat my-resources.json | sensuctl delete
```

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.Read 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			
`auth`	`asset`	`check`	`cluster`
`cluster-role`	`cluster-role-binding`	`entity`	`event`
`filter`	`handler`	`hook`	`mutator`
`namespace`	`role`	`role-binding`	`silenced`
`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 secrets](#)

- ⌐ [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.yml` in `yaml` format or to `my-resources.json` in `wrapped-json` format:

SHELL

```
sensuctl check list --format yaml > my-resources.yml
```

SHELL

```
sensuctl check list --format wrapped-json > my-resources.json
```

To view the definition for a check named `check-cpu`:

SHELL

```
sensuctl check info check-cpu --format yaml
```

SHELL

```
sensuctl check info check-cpu --format wrapped-json
```

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 standard subcommands, 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 review 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

Read the namespaces reference for information about using access control with namespaces.

sensuctl user

Read the [RBAC reference](#) for information about local user management with sensuctl.

sensuctl prune

> **COMMERCIAL FEATURE**: Access sensuctl pruning in the packaged Sensu Go distribution. For more information, read [Get started with commercial features](#).

The `sensuctl prune` subcommand allows you to delete resources that do not appear in a given set of Sensu objects (called a "configuration") from a from a file, URL, or STDIN. For example, you can use `sensuctl create` to to apply a new configuration, then use `sensuctl prune` to prune unneeded resources, resources that were created by a specific user or that include a specific label selector, and more.

> **NOTE**: `sensuctl prune` is an alpha feature and may include breaking changes.
>
> `sensuctl prune` can only delete resources that have the label `sensu.io/managed_by: sensuctl`, which Sensu automatically adds to all resources created with sensuctl. This means you can only use `sensuctl prune` to delete resources that were created with sensuctl.

The pruning operation always follows the role-based access control (RBAC) permissions of the current user. For example, to prune resources in the `dev` namespace, the current user who sends the prune command must have delete access to the `dev` namespace. In addition, pruning requires [cluster-level privileges](#), even when all resources belong to the same namespace.

Supported resource types

> **NOTE**: In Sensu 6.1.0, `sensuctl prune` does not work with hooks.

To retrieve the supported `sensuctl prune` resource types, run:

```
sensuctl describe-type all
```

The response will list all supported `sensuctl prune` resource types:

Fully Qualified Name	Short Name	API Version	Type	Namespaced
authentication/v2.Provider		authentication/v2	Provider	false
licensing/v2.LicenseFile		licensing/v2	LicenseFile	false
store/v1.PostgresConfig		store/v1	PostgresConfig	false
federation/v1.EtcdReplicator		federation/v1	EtcdReplicator	false
secrets/v1.Secret		secrets/v1	Secret	true
secrets/v1.Provider		secrets/v1	Provider	false
searches/v1.Search		searches/v1	Search	true
web/v1.GlobalConfig		web/v1	GlobalConfig	false
core/v2.Namespace	namespaces	core/v2	Namespace	false
core/v2.ClusterRole	clusterroles	core/v2	ClusterRole	false
core/v2.ClusterRoleBinding	clusterrolebindings	core/v2	ClusterRoleBinding	false
core/v2.User	users	core/v2	User	false
core/v2.APIKey	apikeys	core/v2	APIKey	false
core/v2.TessenConfig	tessen	core/v2	TessenConfig	false
core/v2.Asset	assets	core/v2	Asset	true
core/v2.CheckConfig	checks	core/v2	CheckConfig	true
core/v2.Entity	entities	core/v2	Entity	true
core/v2.Event	events	core/v2	Event	true
core/v2.EventFilter	filters	core/v2	EventFilter	true
core/v2.Handler	handlers	core/v2	Handler	true
core/v2.HookConfig	hooks	core/v2	HookConfig	true
core/v2.Mutator	mutators	core/v2	Mutator	true
core/v2.Role	roles	core/v2	Role	true
core/v2.RoleBinding	rolebindings	core/v2	RoleBinding	true
core/v2.Silenced	silenced	core/v2	Silenced	true

NOTE: Short names are only supported for `core/v2` resources.

sensuctl prune flags

Run `sensuctl prune -h` to view command-specific and global flags. The following table describes the command-specific flags.

Flag	Function and important notes
`-a` or `--all-users`	Prunes resources created by all users. Mutually exclusive with the `--`

	`users` flag. Defaults to false.
`-c` or `--cluster-wide`	Prunes any cluster-wide (non-namespaced) resources that are not defined in the configuration. Defaults to false.
`-d` or `--dry-run`	Prints the resources that will be pruned but does not actually delete them. Defaults to false.
`-f` or `--file`	Files, URLs, or directories to prune resources from. Strings.
`-h` or `--help`	Help for the prune command.
`--label-selector`	Prunes only resources that match the specified labels (comma-separated strings). Labels are a commercial feature.
`-r` or `--recursive`	Prune command will follow subdirectories.
`-u` or `--users`	Prunes only resources that were created by the specified users (comma-separated strings). Defaults to the currently configured sensuctl user.

sensuctl prune usage

```
sensuctl prune <resource_type>,<resource_type>... -f <file_or_url> [-r] ... ] --
<namespace> <flags>
```

In this example `sensuctl prune` command:

- ¬ Replace `<resource_type>` with the fully qualified name or short name of the resource you want to prune. You must specify at least one resource type or the `all` qualifier (to prune all resource types).

- ¬ Replace `<file_or_url>` with the name of the file or the URL that contains the set of Sensu objects you want to keep (the configuration).

- ¬ Replace `<namespace>` with the namespace where you want to apply pruning. If you omit the namespace qualifier, the command defaults to the current configured namespace.

- ¬ Replace `<flags>` with the flags you want to use, if any.

Use a comma separator to prune more than one resource in a single command. For example, to prune checks and dynamic runtime assets from the file `checks.yaml` or `checks.json` for the `dev` namespace and the `admin` and `ops` users:

```shell
sensuctl prune core/v2.CheckConfig,core/v2.Asset --file checks.yaml --namespace dev --
users admin,ops
```

SHELL

```shell
sensuctl prune core/v2.CheckConfig,core/v2.Asset --file checks.json --namespace dev --
users admin,ops
```

`sensuctl prune` supports pruning resources by their fully qualified names or short names:

Fully qualified names:

```
sensuctl prune core/v2.CheckConfig,core/v2.Entity
```

Short names:

```
sensuctl prune checks,entities
```

Use the `all` qualifier to prune all supported resources:

```
sensuctl prune all
```

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`

Back up and recover resources with sensuctl

The sensuctl dump command allows you to export your resources to standard out (STDOUT) or to a file. You can export all resources or a subset of them based on a list of resource types. The `dump` command supports exporting in `wrapped-json` and `yaml`.

For example, to export all resources for the current namespace to a file named `my-resources.yml` or `my-resources.json` in `yaml` or `wrapped-json` format for use with sensuctl or `json` format for use with the Sensu API:

SHELL

```
sensuctl dump all --format yaml --file my-resources.yml
```

SHELL

```
sensuctl dump all --format wrapped-json --file my-resources.json
```

SHELL

```
sensuctl dump all --format json --file my-resources.json
```

After you use sensuctl dump to back up your Sensu resources, you can restore them later with sensuctl create.

> **NOTE**: The sensuctl dump command does not export user passwords — you must add the `password_hash` or `password` attribute to any exported users resources before restoring them with sensuctl create. In addition, sensuctl create does not restore API keys from a sensuctl dump backup, although you can use your backup as a reference for granting new API keys.
>
> Because users and API keys require these additional steps to restore with sensuctl create, you might prefer to use the _etcd snapshot and restore process_ as your primary backup and restore method. Take regular etcd snapshots and make regular sensuctl dump backups for extra

This page explains how to back up your resources for two common use cases: before a Sensu version upgrade and to populate new namespaces with existing resources.

Example sensuctl dump commands

To export only checks for only the current namespace to STDOUT in YAML or JSON format:

SHELL

```
sensuctl dump core/v2.CheckConfig --format yaml
```

SHELL

```
sensuctl dump core/v2.CheckConfig --format wrapped-json
```

SHELL

```
sensuctl dump core/v2.CheckConfig --format json
```

To export only handlers and filters for only the current namespace to a file named `my-handlers-and-filters` in YAML or JSON format:

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format yaml --file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format wrapped-json --file my-handlers-and-filters.json
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format json --file my-handlers-
```

```
and-filters.json
```

To export resources for **all namespaces**, add the `--all-namespaces` flag to any sensuctl dump command.For example:

SHELL

```
sensuctl dump all --all-namespaces --format yaml --file my-resources.yml
```

SHELL

```
sensuctl dump all --all-namespaces --format wrapped-json --file my-resources.json
```

SHELL

```
sensuctl dump all --all-namespaces --format json --file my-resources.json
```

SHELL

```
sensuctl dump core/v2.CheckConfig --all-namespaces --format yaml
```

SHELL

```
sensuctl dump core/v2.CheckConfig --all-namespaces --format wrapped-json
```

SHELL

```
sensuctl dump core/v2.CheckConfig --all-namespaces --format json
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --all-namespaces --format yaml --
file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --all-namespaces --format wrapped-
```

```
json --file my-handlers-and-filters.json
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --all-namespaces --format json --
file my-handlers-and-filters.json
```

You can use <u>fully qualified names or short names</u> to specify resources in sensuctl dump commands. Here's an example that uses fully qualified names:

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format yaml --file my-handlers-
and-filters.yml
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format wrapped-json --file my-
handlers-and-filters.json
```

SHELL

```
sensuctl dump core/v2.Handler,core/v2.EventFilter --format json --file my-handlers-
and-filters.json
```

Here's an example that uses short names:

SHELL

```
sensuctl dump handlers,filters --format yaml --file my-handlers-and-filters.yml
```

SHELL

```
sensuctl dump handlers,filters --format wrapped-json --file my-handlers-and-
filters.json
```

SHELL

```
sensuctl dump handlers,filters --format json --file my-handlers-and-filters.json
```

Back up before a Sensu version upgrade

You should create a backup before you upgrade to a new version of Sensu. Here's the step-by-step process:

1. Create a backup folder.

   ```
   mkdir backup
   ```

2. Create a backup of the entire cluster, except entities, events, and role-based access control (RBAC) resources, for all namespaces.

   SHELL

   ```
   sensuctl dump all \
   --all-namespaces \
   --omit
   core/v2.Entity,core/v2.Event,core/v2.APIKey,core/v2.User,core/v2.Role,core/v2.
   RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBinding \
   --format yaml > backup/config.yml
   ```

   SHELL

   ```
   sensuctl dump all \
   --all-namespaces \
   --omit
   core/v2.Entity,core/v2.Event,core/v2.APIKey,core/v2.User,core/v2.Role,core/v2.
   RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBinding \
   --format wrapped-json > backup/config.json
   ```

   SHELL

   ```
   sensuctl dump all \
   ```

```
--all-namespaces \
--omit
core/v2.Entity,core/v2.Event,core/v2.APIKey,core/v2.User,core/v2.Role,core/v2.
RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBinding \
--format json > backup/config.json
```

3. Export your RBAC resources, except API keys and users, for all namespaces.

SHELL

```
sensuctl dump
core/v2.Role,core/v2.RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBindin
g \
--all-namespaces \
--format yaml > backup/rbac.yml
```

SHELL

```
sensuctl dump
core/v2.Role,core/v2.RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBindin
g \
--all-namespaces \
--format wrapped-json > backup/rbac.json
```

SHELL

```
sensuctl dump
core/v2.Role,core/v2.RoleBinding,core/v2.ClusterRole,core/v2.ClusterRoleBindin
g \
--all-namespaces \
--format json > backup/rbac.json
```

4. Export your API keys and users resources for all namespaces.

SHELL

```
sensuctl dump core/v2.APIKey,core/v2.User \
--all-namespaces \
```

```
  --format yaml > backup/cannotrestore.yml
```

```
  sensuctl dump core/v2.APIKey,core/v2.User \
  --all-namespaces \
  --format wrapped-json > backup/cannotrestore.json
```

```
  sensuctl dump core/v2.APIKey,core/v2.User \
  --all-namespaces \
  --format json > backup/cannotrestore.json
```

> **NOTE**: Passwords are not included when you export users. You must add the `password_hash` or `password` attribute to any exported `users` resources before you can use them with `sensuctl create`.
>
> Because users require this additional configuration and API keys cannot be restored from a sensuctl dump backup, consider exporting your API keys and users to a different folder than `backup`.

5. Export your entity resources for all namespaces (if desired).

```
  sensuctl dump core/v2.Entity \
  --all-namespaces \
  --format yaml > backup/inventory.yml
```

```
  sensuctl dump core/v2.Entity \
  --all-namespaces \
  --format wrapped-json > backup/inventory.json
```

```
sensuctl dump core/v2.Entity \
--all-namespaces \
--format json > backup/inventory.json
```

> **NOTE**: If you do not export your entities, proxy check requests will not be scheduled for the excluded proxy entities.

Back up to populate new namespaces

You can create a backup copy of your existing resources with their namespaces stripped from the record. This backup allows you to replicate resources across namespaces without manual editing.

To create a backup of your resources that you can replicate in new namespaces:

1. Create a backup folder.

    ```
    mkdir backup
    ```

2. Back up your pipeline resources for all namespaces, stripping namespaces so that your resources are portable for reuse in any namespace.

    SHELL

    ```
    sensuctl dump
    core/v2.Asset,core/v2.CheckConfig,core/v2.HookConfig,core/v2.EventFilter,core/v2
    .Mutator,core/v2.Handler,core/v2.Silenced,secrets/v1.Secret,secrets/v1.Provide
    r \
    --all-namespaces \
    --format yaml | grep -v "^\s*namespace:" > backup/pipelines.yml
    ```

    SHELL

    ```
    sensuctl dump
    core/v2.Asset,core/v2.CheckConfig,core/v2.HookConfig,core/v2.EventFilter,core/v2
    .Mutator,core/v2.Handler,core/v2.Silenced,secrets/v1.Secret,secrets/v1.Provide
    ```

```
   r \
   --all-namespaces \
   --format wrapped-json | grep -v "^\s*namespace:" > backup/pipelines.json
```

SHELL

```
sensuctl dump
core/v2.Asset,core/v2.CheckConfig,core/v2.HookConfig,core/v2.EventFilter,core/v2
.Mutator,core/v2.Handler,core/v2.Silenced,secrets/v1.Secret,secrets/v1.Provide
r \
--all-namespaces \
--format json | grep -v "^\s*namespace:" > backup/pipelines.json
```

Restore resources from backup

When you are ready to restore your exported resources, use `sensuctl create`.

To restore everything you exported all at once, run:

```
sensuctl create -r -f backup/
```

To restore a subset of your exported resources (in this example, your RBAC resources), run:

SHELL

```
sensuctl create -f backup/rbac.yml
```

SHELL

```
sensuctl create -f backup/rbac.json
```

> **NOTE**: When you export users, required password attributes are not included. You must add a `password_hash` or `password` to `users` resources before restoring them with the `sensuctl create` command.

Supported resource types

NOTE: The `sensuctl describe-type` command deprecates `sensuctl dump --types`.

Use `sensuctl describe-type all` to retrieve the list of supported sensuctl dump resource types.

NOTE: Short names are only supported for `core/v2` resources.

```
sensuctl describe-type all
```

The response will list the names and other details for the supported resource types:

Fully Qualified Name	Short Name	API Version	Type	Namespaced
authentication/v2.Provider		authentication/v2	Provider	false
licensing/v2.LicenseFile		licensing/v2	LicenseFile	false
store/v1.PostgresConfig		store/v1	PostgresConfig	false
federation/v1.EtcdReplicator		federation/v1	EtcdReplicator	false
secrets/v1.Secret		secrets/v1	Secret	true
secrets/v1.Provider		secrets/v1	Provider	false
searches/v1.Search		searches/v1	Search	true
web/v1.GlobalConfig		web/v1	GlobalConfig	false
core/v2.Namespace	namespaces	core/v2	Namespace	false
core/v2.ClusterRole	clusterroles	core/v2	ClusterRole	false
core/v2.ClusterRoleBinding	clusterrolebindings	core/v2	ClusterRoleBinding	false
core/v2.User	users	core/v2	User	false
core/v2.APIKey	apikeys	core/v2	APIKey	false
core/v2.TessenConfig	tessen	core/v2	TessenConfig	false
core/v2.Asset	assets	core/v2	Asset	true
core/v2.CheckConfig	checks	core/v2	CheckConfig	true

core/v2.Entity	entities	core/v2	Entity	true
core/v2.Event	events	core/v2	Event	true
core/v2.EventFilter	filters	core/v2	EventFilter	true
core/v2.Handler	handlers	core/v2	Handler	true
core/v2.HookConfig	hooks	core/v2	HookConfig	true
core/v2.Mutator	mutators	core/v2	Mutator	true
core/v2.Role	roles	core/v2	Role	true
core/v2.RoleBinding	rolebindings	core/v2	RoleBinding	true
core/v2.Silenced	silenced	core/v2	Silenced	true

> **NOTE**: In Sensu 6.1.0, `sensuctl dump` does not work with hooks.

You can also list specific resource types by fully qualified name or short name:

```
sensuctl describe-type core/v2.CheckConfig
```

```
sensuctl describe-type checks
```

To list more than one type, use a comma-separated list:

```
sensuctl describe-type core/v2.CheckConfig,core/v2.EventFilter,core/v2.Handler
```

```
sensuctl describe-type checks,filters,handlers
```

Format the sensuctl describe-type response

Add the `--format` flag to specify how the resources should be formatted in the `sensuctl describe-type` response. The default is unformatted, but you can specify either `wrapped-json` or `yaml`:

SHELL

```
sensuctl describe-type core/v2.CheckConfig --format yaml
```

SHELL

```
sensuctl describe-type core/v2.CheckConfig --format wrapped-json
```

Filter responses with sensuctl

Sensuctl supports response filtering for all [commands using the `list` verb](#). For information about response filtering methods and available label and field selectors, read [API response filtering](#).

Sensuctl-specific syntax

You can use the same methods, selectors, and examples for sensuctl response filtering as for [API response filtering](#), except you'll format your requests with the `--label-selector` and `--field-selector` flags instead of cURL.

The standard sensuctl response filtering syntax is:

```
sensuctl <resource_type> list --<selector> '<filter_statement>'
```

To create a sensuctl response filtering command:

- Replace `<resource_type>` with the resource your filter is based on.
- Replace `<selector>` with either `label-selector` or `field-selector`, depending on which selector you want to use.
- Replace `<filter_statement>` with the filter to apply.

For example:

```
sensuctl event list --field-selector 'linux notin event.entity.subscriptions'
```

Sensuctl response filtering commands will also work with a single equals sign between the selector flag and the filter statement:

```
sensuctl event list --field-selector='linux notin event.entity.subscriptions'
```

The examples demonstrate how to construct sensuctl filter statements for different selectors and operators.

Operators quick reference

Sensuctl response filtering supports two equality-based operators, two set-based operators, one substring matching operator, and one logical operator.

operator	description	example
`==`	Equality	`check.publish == true`
`!=`	Inequality	`check.namespace != "default"`
`in`	Included in	`linux in check.subscriptions`
`notin`	Not included in	`slack notin check.handlers`
`matches`	Substring matching	`check.name matches "linux-"`
`&&`	Logical AND	`check.publish == true && slack in check.handlers`

For details about operators, read about the API response filtering operators.

Examples

Filter responses with label selectors

Use the `--label-selector` flag to filter responses using custom labels.

For example, to return entities with the `proxy_type` label set to `switch`:

```
sensuctl entity list --label-selector 'proxy_type == switch'
```

Filter responses with field selectors

Use the `--field-selector` flag to filter responses using specific resource attributes.

For example, to return entities with the `switches` subscription:

```
sensuctl entity list --field-selector 'switches in entity.subscriptions'
```

To retrieve all events that equal a status of `2` (CRITICAL):

```
sensuctl event list --field-selector 'event.check.status == "2"'
```

To retrieve all entities whose name includes the substring `webserver` :

```
sensuctl entity list --field-selector 'entity.name matches "webserver"'
```

Use the logical AND operator

To use the logical AND operator (`&&`) to return checks that include a `linux` subscription in the `dev` namespace:

```
sensuctl check list --field-selector 'linux in check.subscriptions && dev in check.namespace'
```

Combine label and field selectors

You can combine the `--label-selector` and `--field-selector` flags in a single command.

For example, this command returns checks with the `region` label set to `us-west-1` that also use the `slack` handler:

```
sensuctl check list --label-selector 'region == "us-west-1"' --field-selector 'slack
in check.handlers'
```

Set environment variables with sensuctl

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 (for example, JSON,
YAML, etc.)
SENSU_ACCESS_TOKEN                Current API access token in sensuctl
SENSU_ACCESS_TOKEN_EXPIRES_AT     Timestamp specifying when the current API access
token expires
SENSU_REFRESH_TOKEN               Refresh token used to obtain a new access token
SENSU_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
```

These examples demonstrate how to use sensuctl to export and set environment variables and configure your shell:

BASH

```
export SENSU_API_URL="http://127.0.0.1:8080"
export SENSU_NAMESPACE="default"
export SENSU_FORMAT="tabular"
export SENSU_ACCESS_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
export SENSU_ACCESS_TOKEN_EXPIRES_AT="1567716187"
export SENSU_REFRESH_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
export SENSU_TRUSTED_CA_FILE=""
export SENSU_INSECURE_SKIP_TLS_VERIFY="true"
eval $(sensuctl env)
```

CMD

```
SET SENSU_API_URL=http://127.0.0.1:8080
SET SENSU_NAMESPACE=default
```

```
SET SENSU_FORMAT=tabular
SET SENSU_ACCESS_TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x
SET SENSU_ACCESS_TOKEN_EXPIRES_AT=1567716676
SET SENSU_REFRESH_TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x
SET SENSU_TRUSTED_CA_FILE=
SET SENSU_INSECURE_SKIP_TLS_VERIFY=true
@FOR /f "tokens=*" %i IN ('sensuctl env --shell cmd') DO @%i
```

POWERSHELL

```
$Env:SENSU_API_URL = "http://127.0.0.1:8080"
$Env:SENSU_NAMESPACE = "default"
$Env:SENSU_FORMAT = "tabular"
$Env:SENSU_ACCESS_TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
$Env:SENSU_ACCESS_TOKEN_EXPIRES_AT = "1567716738"
$Env:SENSU_REFRESH_TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.x.x"
$Env:SENSU_TRUSTED_CA_FILE = ""
$Env:SENSU_INSECURE_SKIP_TLS_VERIFY = "true"
& sensuctl env --shell powershell | Invoke-Expression
```

Use sensuctl with Bonsai

Sensuctl supports installing dynamic runtime asset definitions directly from <u>Bonsai, the Sensu asset hub</u>, and checking your Sensu backend for outdated dynamic runtime assets. You can also use `sensuctl command` to install, execute, list, and delete commands from Bonsai or a URL.

Install dynamic runtime asset definitions

To install a dynamic runtime asset definition directly from Bonsai, use `sensuctl asset add <namespace/name>:<version>`. Replace `<namespace/name>` with the namespace and name of the dynamic runtime asset from Bonsai and `<version>` with the asset version you want to install.

To automatically install the latest version of the plugin, you do not need to specify the version: `sensuctl asset add <namespace/name>`.

> **NOTE**: We recommend specifying the asset version you want to install to maintain the stability of your observability infrastructure. If you do not specify a version to install, Sensu automatically installs the latest version, which may include breaking changes.
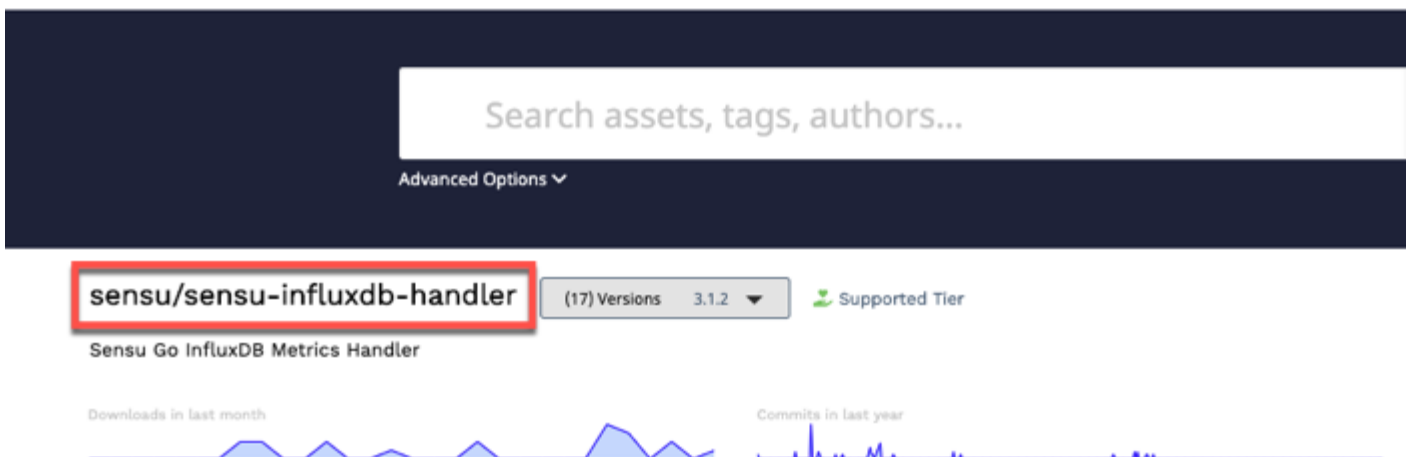
For example, to install version 3.1.1 of the <u>Sensu InfluxDB Handler</u> dynamic runtime asset:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.1.1
```

The response should be similar to this example:

```
fetching bonsai asset: sensu/sensu-influxdb-handler:3.1.1
added asset: sensu/sensu-influxdb-handler:3.1.1

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["sensu/sensu-influxdb-handler"].
```

You can also use the `--rename` flag to rename the dynamic runtime asset on install:

```
sensuctl asset add sensu/sensu-influxdb-handler:3.1.1 --rename influxdb-handler
```

> **NOTE**: Sensu does not download and install dynamic runtime asset builds onto the system until
> they are needed for command execution.Read _the asset reference_ for more information about
> dynamic runtime asset builds.

Check your Sensu backend for outdated dynamic runtime assets

To check your Sensu backend for dynamic runtime assets that have newer versions available on
Bonsai, use `sensuctl asset outdated`.This will print a list of dynamic runtime assets installed in the
backend whose version is older than the newest version available on Bonsai:

```
sensuctl asset outdated
```

The response should be similar to this example:

```
          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.

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 (for example, https://path/to/asset.tar.gz).Replace `<archive_checksum>` with the checksum you want to use.Replace `<alias>` with a unique name for the command.

Replace `<flags>` with the flags you want to use.Run `sensuctl command install -h` to view flags. Flags are optional and apply only to the `install` command — they are not saved as part of the command you are installing.

For example, to install a command-test dynamic runtime asset via URL with no flags:

```
sensuctl command install command-test --url https://github.com/amdprophet/command-
test/releases/download/v0.0.4/command-test_0.0.4_darwin_amd64.tar.gz --checksum
8b15a170e091dab42256fe64ca7c4a050ed49a9dbfd6c8129c95506a8a9a91f2762ac1a6d24f4fc54543
0613fd45abc91d3e5d3605fcfffb270dcf01996caa7f
```

> **NOTE**: Dynamic runtime asset definitions with multiple asset builds are only supported via Bonsai.

Execute commands

To execute a sensuctl command plugin via its dynamic runtime asset's bin/entrypoint executable:

```
sensuctl command exec <alias> <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 <command> <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.

Web UI

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

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

The web UI homepage provides a high-level overview of the overall health of the systems under Sensu's management, with a summary of active incidents, the number of incidents by severity, the types of entities under management, and the numbers of entities and incidents per namespace.
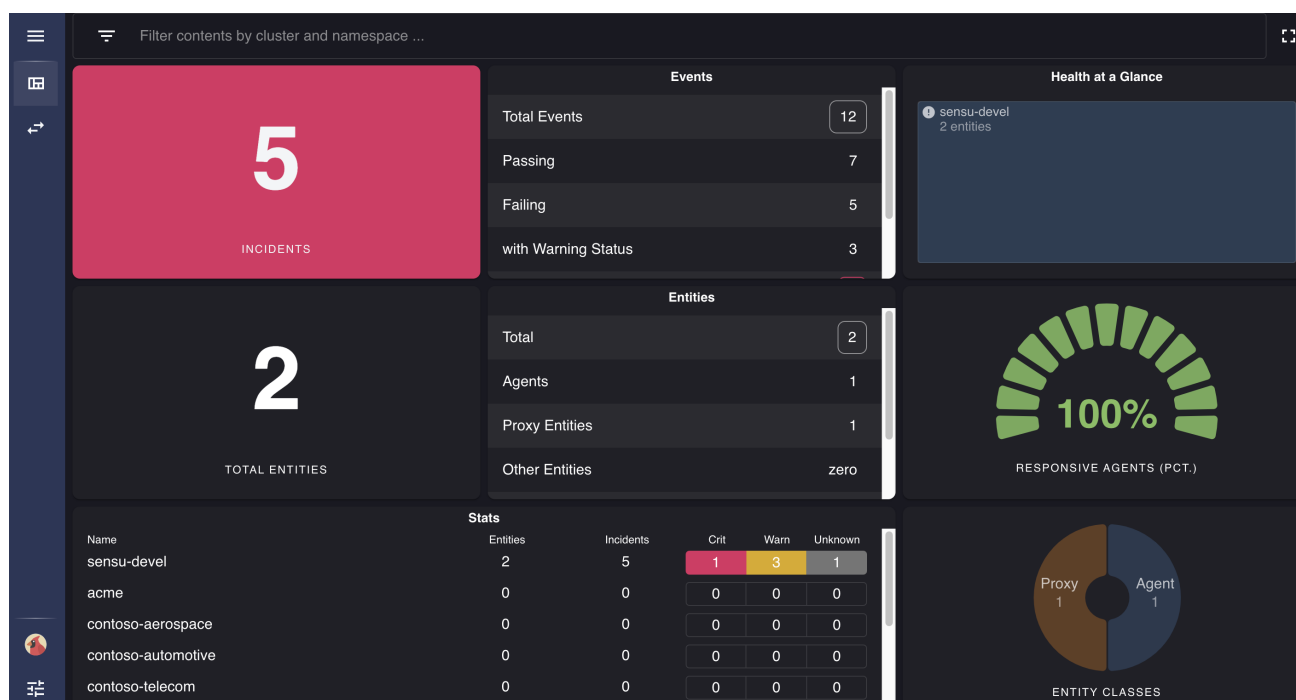
Access the web UI

After you [start the Sensu backend](#), you can access the web UI in your browser by visiting http://localhost:3000.

NOTE: *You may need to replace* `localhost` *with the hostname or IP address where the Sensu backend is running.*

Sign in to the web UI

Sign in to the web UI with the username and password you used to configure sensuctl.

The web UI uses your username and password to obtain access and refresh tokens via the Sensu authentication API. The access and refresh tokens are JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature. The access and refresh tokens are saved in your browser's local storage.

The web UI complies with Sensu role-based access control (RBAC), so individual users can view information according to their access configurations. Read the RBAC reference for default user credentials and instructions for creating new users.

Change web UI themes

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

View and manage resources in the web UI

You can view and manage Sensu resources in the web UI, including events, entities, silences, checks, handlers, event filters, and mutators.

Use the namespace switcher

Beyond the homepage, the web UI displays events, entities, and resources for a single namespace at a time.By default, the web UI displays the `default` namespace.

To switch namespaces, select the menu icon in the upper-left corner or press the `Ctrl+K` keyboard shortcut and choose a namespace from the dropdown.

> **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, read Get started with commercial features.
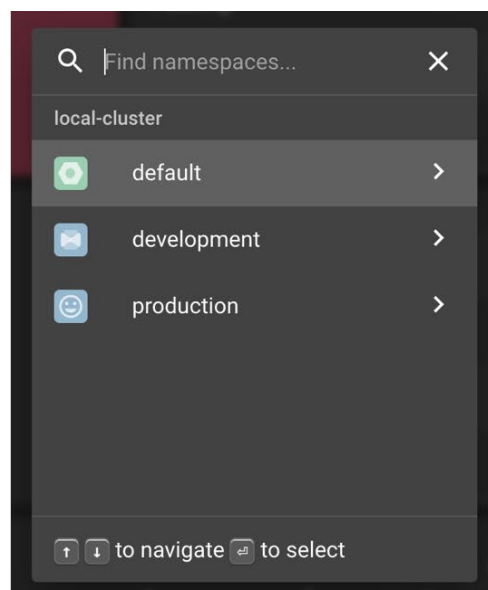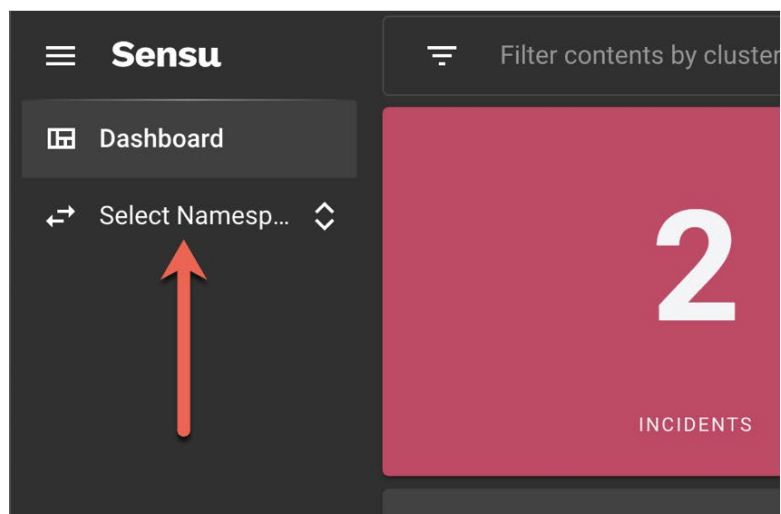
When you switch to a namespace, the left navigation menu loads so you can select specific pages for events, entities, silences, and configuration, which includes checks, handlers, event filters, and mutators:
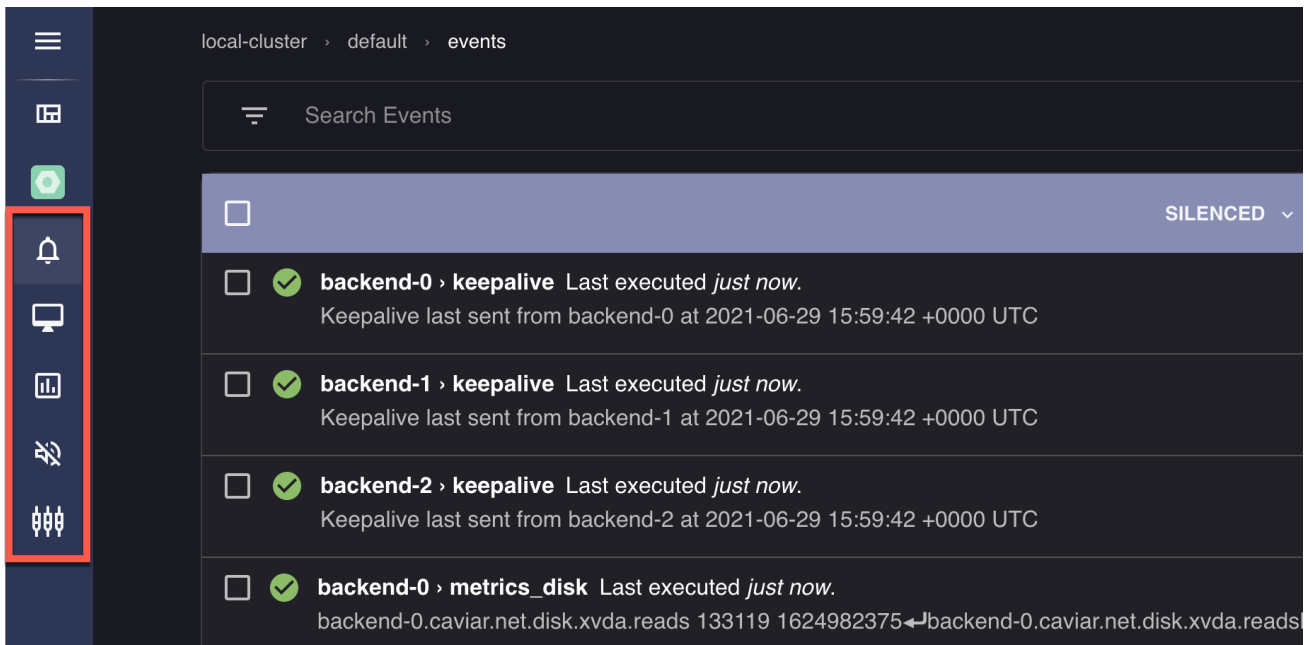
Click the triple-bar icon at the top of the left-navigation menu to expand the menu and display page labels:
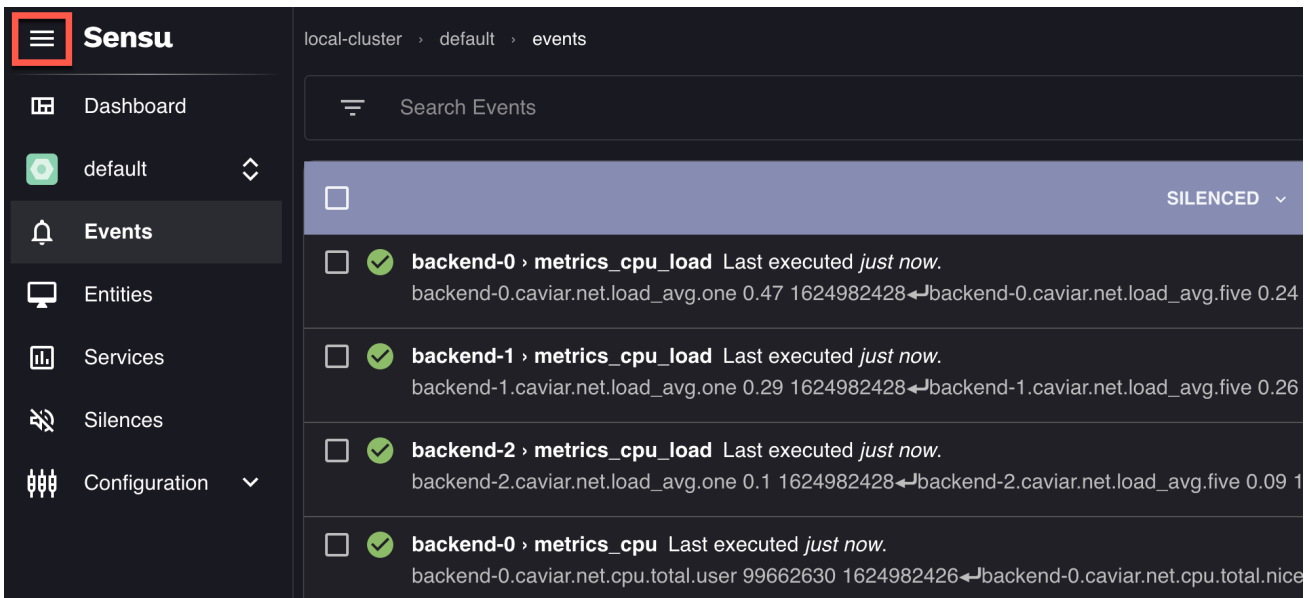
Manage events

The Events page opens by default when you navigate to a namespace, with an automatic filter to show only events with a non-passing status (i.e. `event.check.state != passing`):
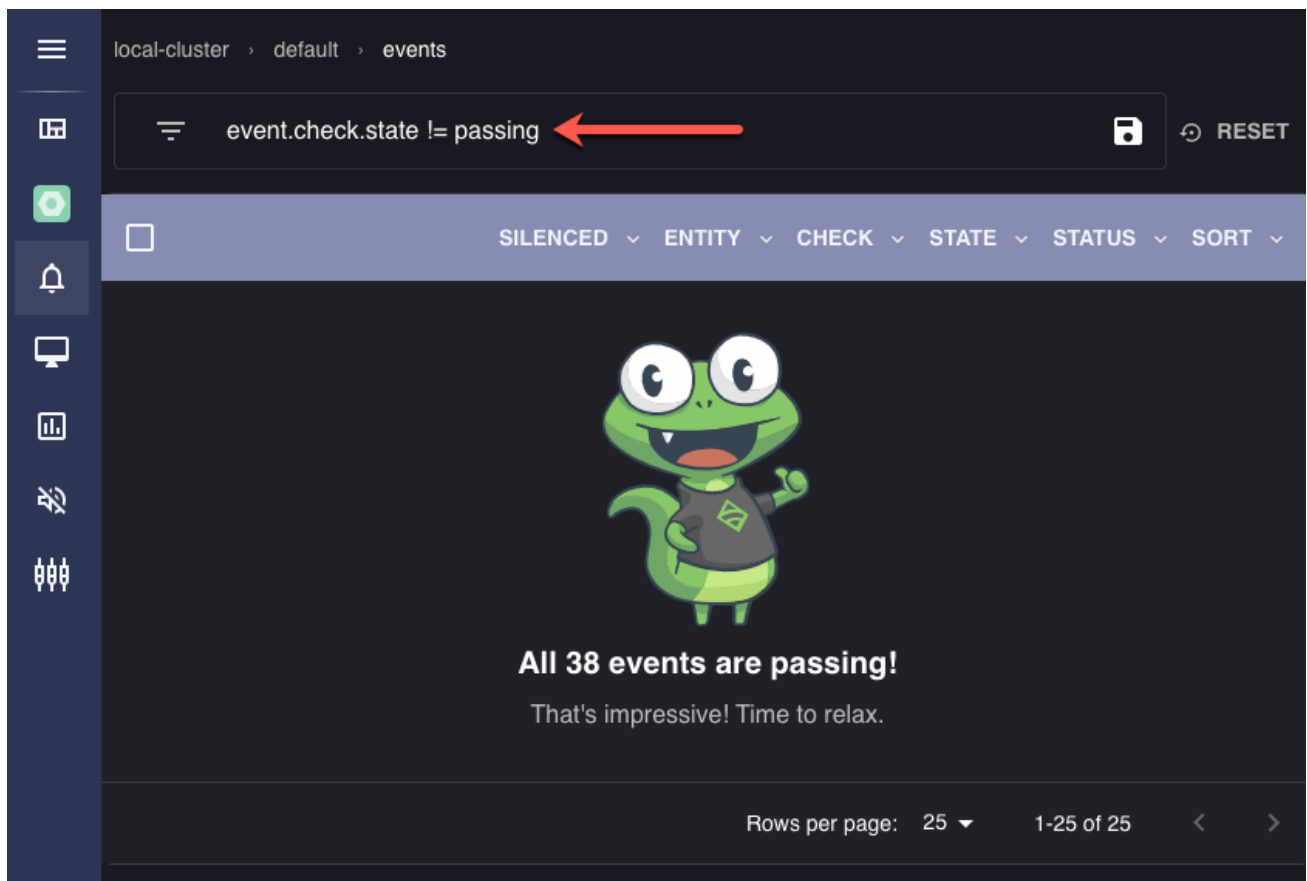
The top row of the events list includes several other options for filtering and sorting events:
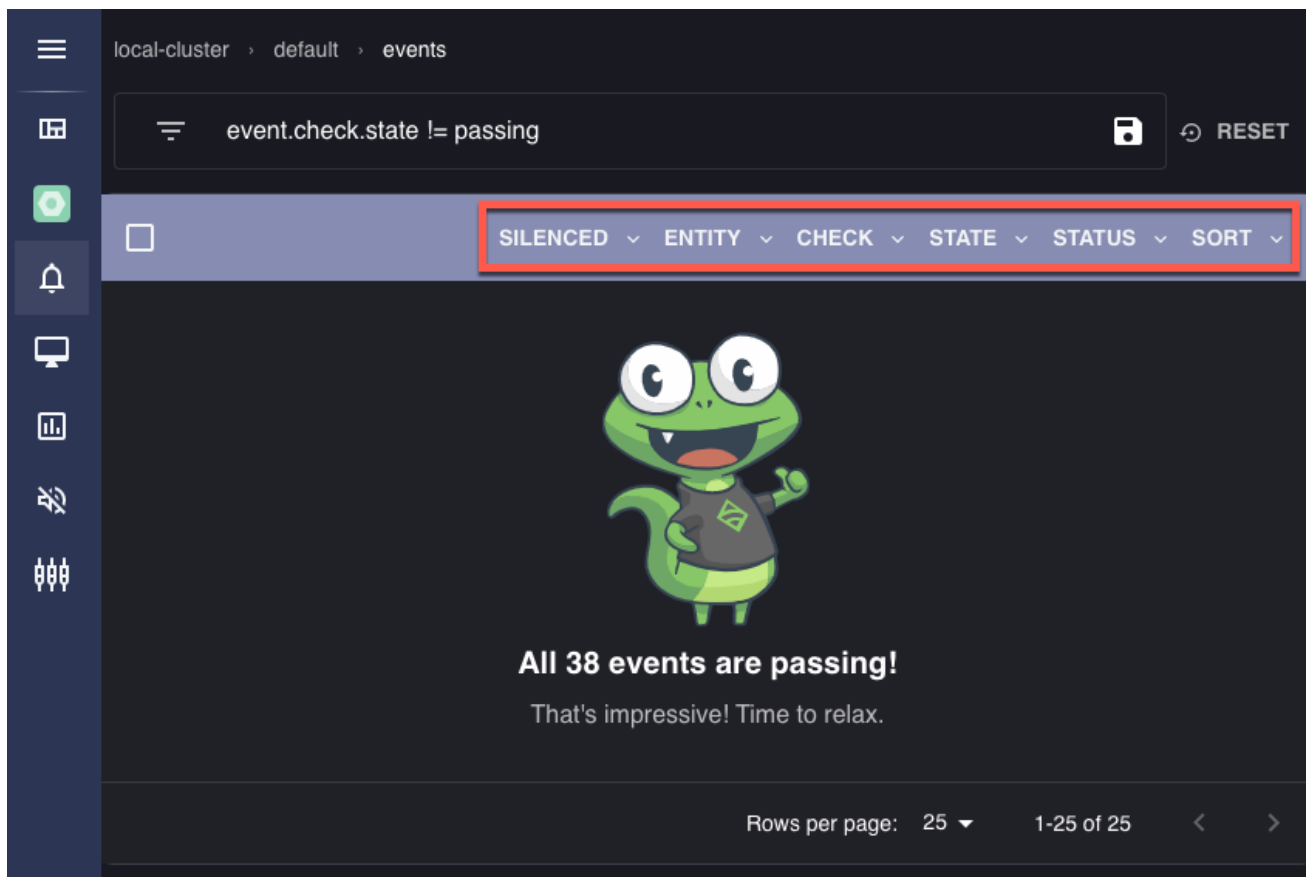
Click the check boxes to select one or more events and resolve, silence, or delete them directly from the Events page:
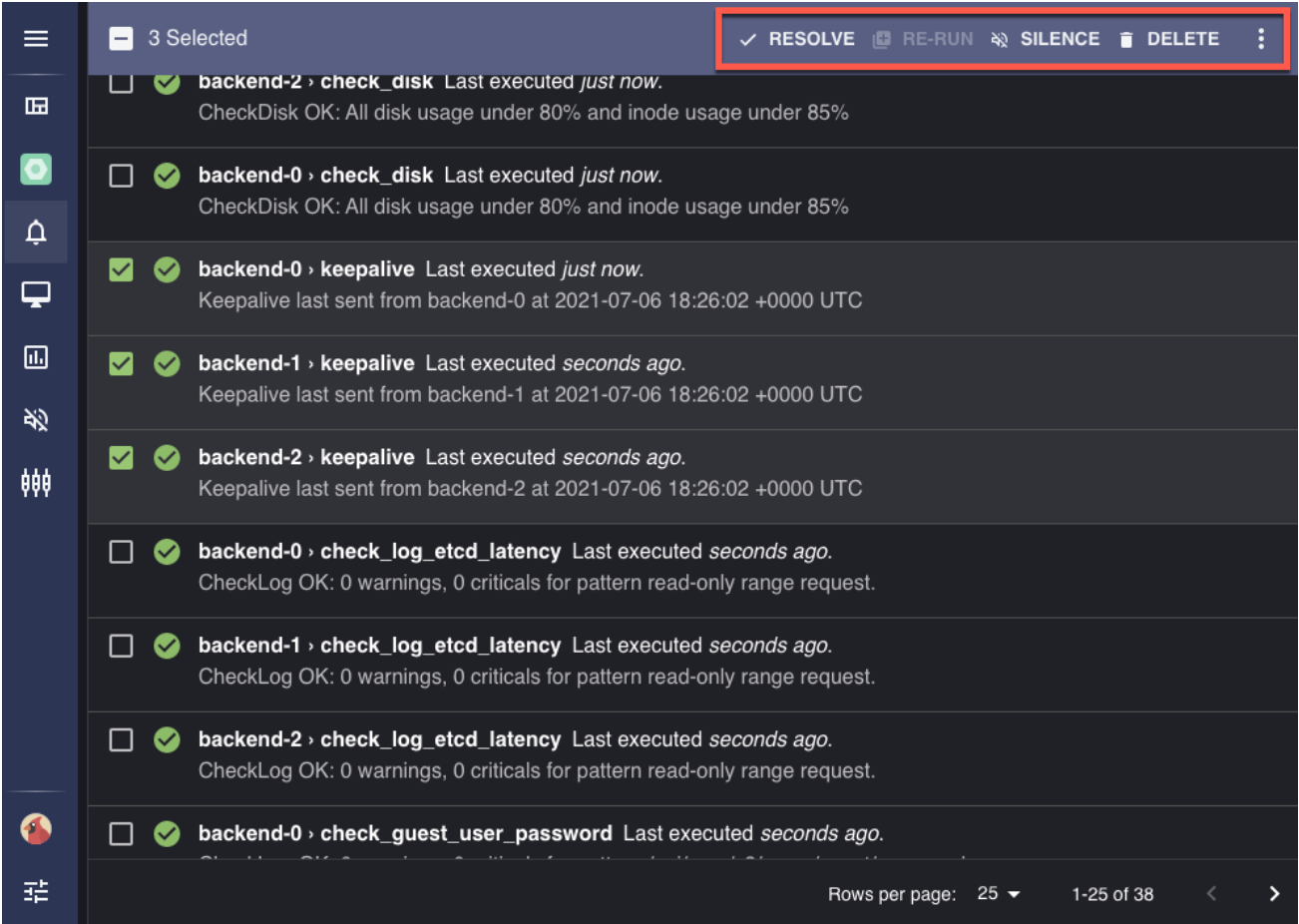
Click an event name to view details like status, output, number of occurrences, labels and annotations, related check configuration (if the event was produced by a service check), and associated entity, as well as a timeline that displays the event's last 20 statuses at a glance:
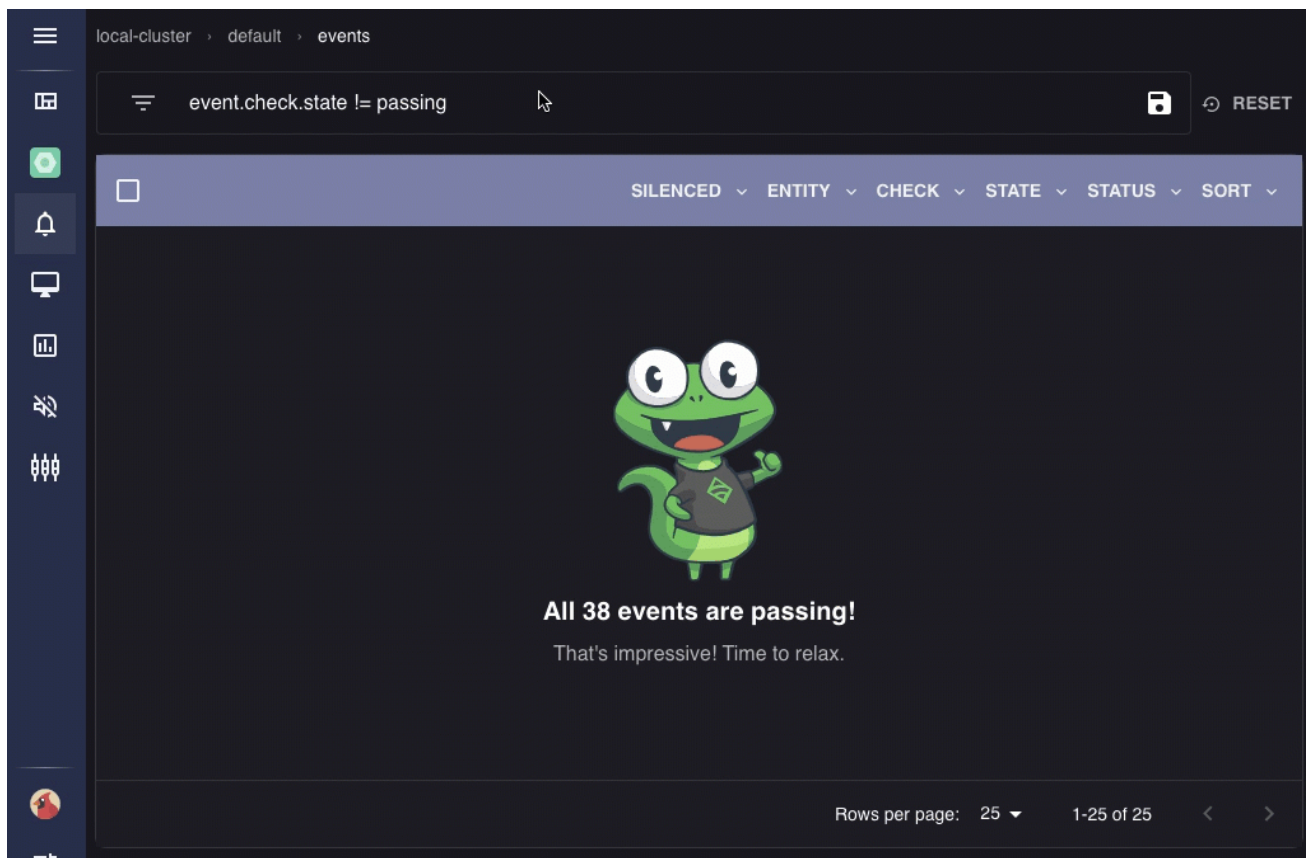
Manage entities

The Entities page provides real-time inventory information for the namespace's endpoints under Sensu management.The top row of the entities list includes options for filtering and sorting entities on the page:

Click the check boxes to select one or more entities and silence or delete them directly from the Entities page:

Click an entity name to view details about associated events, system properties, and labels and annotations:

Manage silences

Create silences by check or subscription name and clear silences in the web UI Silences page. The Silences page lists all active silences for the namespace. The top row of the silences list includes options for filtering and sorting silences on the page:

Click + NEW to open a modal window and create silences for individual events, by check or subscription name, or by entity:
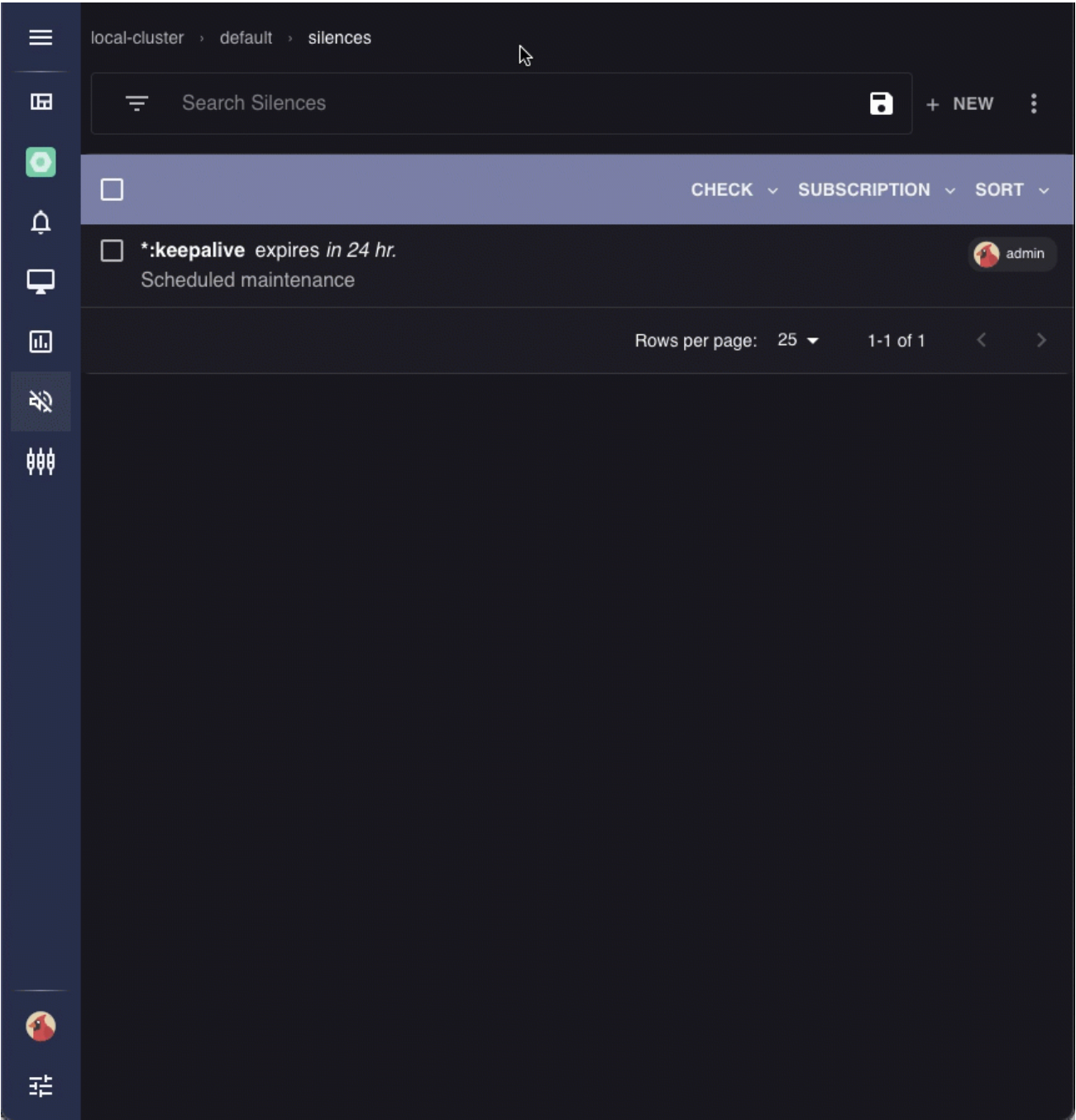
You can also silence individual checks and entities from their detail pages in the web UI.

After you create a silence, it will be listed in the web UI Silences page until you clear the silence or the silence expires.

Manage configuration resources

Under the Configuration menu option, you can access check, handler, event filter, and mutator resources. Each resource page lists the namespace's resources. The top row of each page includes options for filtering and sorting the listed resources.
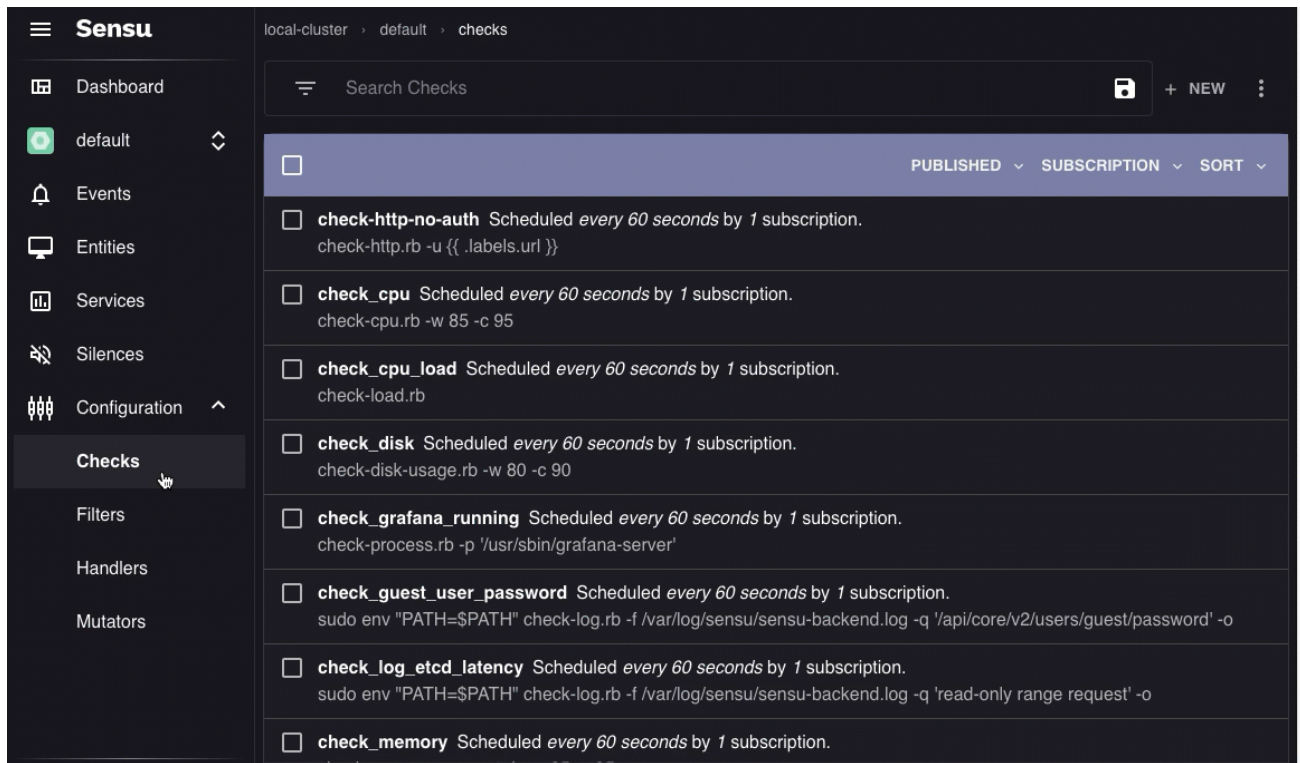
Click a resource name to view detailed information and edit or delete it.

On the Checks page, click the check boxes to select one or more checks to execute, silence, unpublish, or delete them. You can also execute individual checks on demand from their check detail pages to test your observability pipeline:

Search in the web UI

The Sensu web UI includes basic search and filtering functions you can use to build customized views of your Sensu resources.Sensu also supports advanced web UI searches based on a wider range of resource attributes and custom labels as a commercial feature.

When you apply a search to a web UI page, it creates a unique link for the page of search results.You can bookmark these links and share your favorite search combinations.You can also save your favorite searches.

Search operators

Web UI search supports two equality-based operators, two set-based operators, one substring matching operator, and one logical operator.

operator	description	example
`==`	Equality	`check.publish == "true"`
`!=`	Inequality	`check.namespace != "default"`
`in`	Included in	`"linux" in check.subscriptions`
`notin`	Not included in	`"slack" notin check.handlers`
`matches`	Substring matching	`check.name matches "linux-"`
`&&`	Logical AND	`check.publish == "true" && "slack" in check.handlers`

For details about operators, read about the API response filtering operators.

Use quick search

The web UI quick search allows you to query and filter Sensu resources without using search syntax. Type your search term into the search field on any page of the web UI and press `Enter` .Sensu will auto-complete a simple search statement for the resources on that page using substring matching.

For example, on the Events page in the web UI, if you type `mysql` into the search field, Sensu will auto-complete the search statement to `event.check.name matches "mysql"` .

Create basic searches

Sensu includes these basic search functions:

- **Events page**: search by entity, check, status, and silenced/unsilenced.

- **Entities page**: search by entity class and subscription.

- **Silences page**: search by check and subscription.

- **Checks page**: search by subscription and published/unpublished.

- **Handlers page**: search by handler type.

- **Filters page**: search by action.

If you are using the basic web UI search functions, you can create a search by clicking in the search bar at the top of the web UI page:

1. In the web UI, open the page of resources you want to search.
2. Click in the search bar at the top of the web UI page.
3. Select the attribute you want to search for from the dropdown list of options.
4. Click in the search bar again and select the search to apply.
5. Press **Return/Enter**.

> **NOTE**: You do not need to specify a resource type in web UI search because you must navigate to the resource page before you construct the search.

Create advanced searches

> **COMMERCIAL FEATURE**: Access advanced web UI searches in the packaged Sensu Go distribution.For more information, read Get started with commercial features.

Sensu supports advanced web UI searches using a wider range of attributes, including custom labels. You can use the same methods, fields, and examples for web UI searches as for API response filtering, with some syntax differences.

To search resources based on fields and labels, you'll write a brief search statement. Depending on the operator you're using, the web UI search syntax is either:

```
SEARCH_TERM OPERATOR FIELD
```

or

```
FIELD OPERATOR SEARCH_TERM
```

Fields are specific resource attributes in dot notation. For example, this search will retrieve all events for entities with the `linux` subscription:

```
"linux" in event.entity.subscriptions
```

This search will retrieve all events that whose status is *not* equal to `passing`:

```
event.check.state != "passing"
```

To display only events for checks with the subscription `webserver`, enter this search statement on the **Events page**:

```
"webserver" in event.check.subscriptions
```

To display only checks that use the `slack` handler, enter this search statement on the **Checks page**:

```
"slack" in check.handlers
```

Search for numbers or special characters

If you are searching for a value that begins with a number, place the value in single or double quotes:

```
entity.name == '1b04994n'
entity.name == "1b04994n"
```

Likewise, to search string values that include special characters like hyphens and underscores, place the value in single or double quotes:

```
entity.labels.region == 'us-west-1'
entity.labels.region == "us-west-1"
```

To display only events at `2` (CRITICAL) status:

```
event.check.status == "2"
```

Search for labels

Labels are treated like any other field in web UI searches.

For example, to search based on a check label `version`, use:

```
check.labels.version matches "7"
```

To display only checks with the `type` label set to `server`, enter this search statement on the **Checks page**:

```
check.labels.type == "server"
```

To search for entities that are labeled for any region in the US (for example, `us-east-1`, `us-west-1`, and so on):

```
entity.labels.region matches "us"
```

> **NOTE**: Web UI searches for label names that include hyphens are not supported. Searches that include a hyphenated label name, such as `entity.labels.imported-by`, will return an unsupported token error.

Search for event labels

For label-based event searches, the web UI merges check and entity labels into a single search term: `event.labels.[KEY]`.

For example, to display events with the `type` label set to `server`, enter this search statement on the **Events** page:

```
event.labels.type == "server"
```

This search will retrieve events with the `type` label set to `server`, no matter whether the label is defined in the event's corresponding check or entity configuration.

Use the logical AND operator

To use the logical AND operator (`&&`) to return checks that include a `linux` subscription and the `slack` handler:

```
"linux" in check.subscriptions && "slack" in check.handlers
```

To return events that include a `windows` check subscription and any email handler:

```
"windows" in event.check.subscriptions && event.check.handlers matches "email"
```

Save a search

> **COMMERCIAL FEATURE**: Access saved web UI searches in the packaged Sensu Go distribution. For more information, read *Get started with commercial features*.

To save a web UI search:

1. Create a web UI search.
2. Click the save icon at the right side of the search bar:
3. Click **Save this search**.
4. Type the name you want to use for the saved search.
5. Press **Return/Enter**.

Sensu saves your web UI searches to etcd in a namespaced resource named `searches` . To recall a saved web UI search, a Sensu user must be assigned to a role that includes permissions for both the `searches` resource and the namespace where you save the search.

The role-based access control (RBAC) reference includes example workflows that demonstrate how to configure a user's roles and role bindings to include full permissions for namespaced resources, including saved searches.

Recall a saved search

To recall a saved search, click the save icon in the search bar and select the name of the search you want to recall.

You can combine an existing saved search with a new search to create a new saved search. To do this, recall a saved search, add the new search statement in the search bar, and save the combination as a new saved search.

Delete a saved search

To delete a saved search:

1. Click the save icon in the search bar:
2. Click the delete icon next to the search you want to delete:

Use the sort function

Use the **SORT** dropdown menu to sort search results. You can sort all resources by name, but events and silences have additional sorting options:

- **Events page**: sort by last OK, severity, newest, and oldest.

- **Silences page**: sort by start date.

Configure the web UI

Web UI configuration allows you to define certain display options for the Sensu web UI, such as which web UI theme to use, the number of items to list on each page, and which URLs and linked images to expand. You can define a single custom web UI configuration to federate to all, some, or only one of your clusters.

Create a web UI configuration

Use the web UI configuration API or sensuctl create to create a `GlobalConfig` resource. The web UI configuration reference describes each attribute you can configure in the `GlobalConfig` resource.

> **_NOTE_**_: Each cluster should have only one web configuration._

If an individual user's settings conflict with the web UI configuration settings, Sensu will use the individual user's settings. For example, if a user's system is set to dark mode and their web UI settings are configured to use their system settings, the web UI will use dark mode for that user, even if you set the theme to `classic` in your web UI configuration.

Federate a web UI configuration to specific clusters

The web UI configuration in use is provided by the cluster you are connected to. For example, if you open the web UI for https://cluster-a.sensu.my.org:3000, the web UI display will be configured according to the `GlobalConfig` resource for cluster-a.

In a federated environment, you can create an etcd replicator for your `GlobalConfig` resource so you can use it for different clusters:

YML

```yaml
---
type: EtcdReplicator
api_version: federation/v1
metadata:
  name: web_global_config
spec:
  api_version: web/v1
  ca_cert: /path/to/ssl/trusted-certificate-authorities.pem
  cert: /path/to/ssl/cert.pem
  insecure: false
  key: /path/to/ssl/key.pem
  replication_interval_seconds: 120
  resource: GlobalConfig
  url: "http://127.0.0.1:2379"
```

JSON

```json
{
  "type": "EtcdReplicator",
  "api_version": "federation/v1",
  "metadata": {
    "name": "web_global_config"
  },
  "spec": {
    "api_version": "web/v1",
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "insecure": false,
    "key": "/path/to/ssl/key.pem",
    "replication_interval_seconds": 120,
    "resource": "GlobalConfig",
    "url": "http://127.0.0.1:2379"
  }
}
```

Debugging in federated environments

In a federated environment, a problem like incorrect configuration, an error, or a network issue could prevent a cluster from appearing in the web UI namespace switcher.

If you set the `always_show_local_cluster` attribute to `true` in your web UI configuration, the namespace switcher will display a heading for each federated cluster, along with the local-cluster heading to indicate the cluster you are currently connected to.With `always_show_local_cluster` set to `true`, the cluster administrator can directly connect to the local cluster even if there is a problem that would otherwise prevent the cluster from being listed in the namespace switcher.

> **NOTE**: Use the `always_show_local_cluster` attribute only in federated environments.In a single-cluster environment, the namespace switcher will only list a local-cluster heading and the namespaces for that cluster.

Searches reference

With the saved searches feature, you can apply search parameters to your entities, events, and resources and save them to etcd in a namespaced resource named `searches`.

The saved searches feature is designed to be used directly in the web UI.However, you can create, retrieve, update, and delete saved searches with the searches API.

Search for events with any status except passing

The following saved search will retrieve all events that have any status except `passing`:

YML

```yaml
---
type: Search
api_version: searches/v1
metadata:
  name: events-not-passing
  namespace: default
spec:
  parameters:
  - status:incident
  - status:warning
  - status:critical
  - status:unknown
  resource: core.v2/Event
```

JSON

```json
{
  "type": "Search",
```

```json
  "api_version": "searches/v1",
  "metadata": {
    "name": "events-not-passing",
    "namespace": "default"
  },
  "spec": {
    "parameters": [
      "status:incident",
      "status:warning",
      "status:critical",
      "status:unknown"
    ],
    "resource": "core.v2/Event"
  }
}
```

Search for published checks with a specific subscription and region

The following saved search will retrieve all published checks for the `us-west-1` region with the `linux` subscription:

YML

```yaml
---
type: Search
api_version: searches/v1
metadata:
  name: published-checks-linux-uswest
  namespace: default
spec:
  parameters:
  - published:true
  - subscription:linux
  - 'labelSelector: region == "us-west-1"'
  resource: core.v2/CheckConfig
```

JSON

```
{
  "type": "Search",
  "api_version": "searches/v1",
  "metadata": {
    "name": "published-checks-linux-uswest",
    "namespace": "default"
  },
  "spec": {
    "parameters": [
      "published:true",
      "subscription:linux",
      "labelSelector: region == \"us-west-1\""
    ],
    "resource": "core.v2/CheckConfig"
  }
}
```

Searches specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Searches should always be type `Search`.
required	Required for search entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

YML

example	

```yaml
type: Search
```

JSON

```json
{
  "type": "Search"
}
```

```
    }
```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For searches in this version of Sensu, the `api_version` should always be `searches/v1`.
required	Required for search entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String

example	**YML**
| | ```
api_version: searches/v1
``` |
| | **JSON** |
| | ```
{
  "api_version": "searches/v1"
}
``` |

metadata

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

YML

example	
| | ```
metadata:
  name: us-west-server-incidents
  namespace: default
``` |

JSON

```json
{
  "metadata": {
    "name": "us-west-server-incidents",
    "namespace": "default"
  }
}
```

spec

description	Top-level map that includes the search spec attributes. The spec contents will depend on the search parameters you apply and save.
required	Required for silences in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs **YML**
| example | ```
spec:
  parameters:
  - entity:server-testing
  - check:server-health
  - status:incident
  - labelSelector:region == "us-west-1"
  resource: core.v2/Event
``` |

JSON

```json
{
  "spec": {
    "parameters": [
```

```
        "entity:server-testing",
        "check:server-health",
        "status:incident",
        "labelSelector:region == \"us-west-1\""
      ],
      "resource": "core.v2/Event"
    }
  }
```

Metadata attributes

name	
description	Search identifier generated from the combination of a subscription name and check name.
required	true
type	String
example	**YML**

```
  name: us-west-server-incidents
```

JSON

```
  {
    "name": "us-west-server-incidents"
  }
```

namespace	
description	Sensu RBAC namespace that the search belongs to.
required	false

type	String
default	`default`

YML

example	

```yml
namespace: default
```

JSON

```json
{
  "namespace": "default"
}
```

Spec attributes

parameters	
description	Parameters the search will apply.
required	true
type	Array

YML

example	

```yml
parameters:
- entity:server-testing
- check:server-health
- status:incident
- labelSelector:region == "us-west-1"
```

JSON

```json
{
  "parameters": [
    "entity:server-testing",
    "check:server-health",
```

```
      "status:incident",
      "labelSelector:region == \"us-west-1\""
    ]
  }
```

resource

description	Fully qualified name of the resource included in the search.
required	true
type	String

YML

```
resource: core.v2/Event
```

JSON

```
{
  "resource": "core.v2/Event"
}
```

Parameters

action

description	For event filter searches, the type of filter to include in the search: `allow` or `deny`.
required	false
type	String

YML

example	

```yaml
parameters:
- action:allow
```

JSON

```json
{
  "parameters": [
    "action:allow"
  ]
}
```

check

description	Name of the check to include in the search.
required	false
type	String

YML

example	

```yaml
parameters:
- check:server-health
```

JSON

```json
{
  "parameters": [
    "check:server-health"
  ]
}
```

class

description	For entity searches, the entity class to include in the search: `agent` or

`proxy` .

required	false
type	String

example	**YML**

```yml
parameters:
- class:agent
```

JSON

```json
{
  "parameters": [
    "class:agent"
  ]
}
```

entity

description	Name of the entity to include in the search.
required	false
type	String

example	**YML**

```yml
parameters:
- entity:server-testing
```

JSON

```json
{
  "parameters": [
    "entity:server-testing"
  ]
}
```

event

description	Name of the event to include in the search.
required	false
type	String

example

YML

```yaml
parameters:
- event:server-testing
```

JSON

```json
{
  "parameters": [
    "event:server-testing"
  ]
}
```

published

description	If `true`, the search will include only published resources. Otherwise, `false`.
required	false
type	Boolean

example

YML

```yaml
parameters:
- published:true
```

JSON

```json
{
  "parameters": [
    "published:true"
  ]
}
```

silenced

description	If `true` , the search will include only silenced events. Otherwise, `false` .
required	false
type	Boolean

example

YML

```yaml
parameters:
- silenced:true
```

JSON

```json
{
  "parameters": [
    "silenced:true"
  ]
}
```

status

description	Status of the events, entities, or resources to include in the search.
required	false
type	String

YML

example	
| | ```yaml
parameters:
- status:incident
``` |
| | **JSON** |
| | ```json
{
  "parameters": [
    "status:incident"
  ]
}
``` |

subscription

description	Name of the subscription to include in the search.
required	false
type	String
	YML
| example | ```yaml
parameters:
- subscription:web
``` |
| | **JSON** |
| | ```json
{
  "parameters": [
    "subscription:web"
  ]
}
``` |

type

description	For handler searches, the type of hander to include in the search: `pipe`, `set`, `tcp`, or `udp`.
required	false
type	String
example	**YML**

```yml
parameters:
- type:pipe
```

JSON

```json
{
  "parameters": [
    "type:pipe"
  ]
}
```

Web UI configuration reference

Web UI configuration allows you to define certain display options for the Sensu web UI, such as which web UI theme to use, the number of items to list on each page, and which URLs and linked images to expand. You can define a single custom web UI configuration to federate to all, some, or only one of your clusters.

> **NOTE**: *Each cluster should have only one web configuration.*

Web UI configuration example

In this web UI configuration example:

- ¬ Details for the local cluster will not be displayed

- ¬ Each page will list 50 items

- ¬ The web UI will use the classic theme

YML ¬ Expanded links and images will be allowed for the listed URLs

```
type: GlobalConfig
api_version: web/v1
metadata:
  name: custom-web-ui
spec:
  always_show_local_cluster: false
  default_preferences:
    page_size: 50
    theme: classic
  link_policy:
    allow_list: true
    urls:
```

```
    - https://example.com
    - steamapp://34234234
    - //google.com
    - //*.google.com
    - //bob.local
    - https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic
```

JSON

```json
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui",
    "created_by": "admin"
  },
  "spec": {
    "always_show_local_cluster": false,
    "default_preferences": {
      "page_size": 50,
      "theme": "classic"
    },
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
      ]
    }
  }
}
```

Web UI configuration specification

Top-level attributes

type

description	Top-level attribute that specifies the resource type. For web UI configuration, the type should always be `GlobalConfig`.
required	Required for web UI configuration in `wrapped-json` or `yaml` format.
type	String

example

YML

```yml
type: GlobalConfig
```

JSON

```json
{
  "type": "GlobalConfig"
}
```

api_version

description	Top-level attribute that specifies the Sensu API group and version. For web UI configuration in this version of Sensu, the api_version should always be `web/v1`.
required	Required for web UI configuration in `wrapped-json` or `yaml` format.
type	String

example

YML

```yml
api_version: web/v1
```

JSON

```json
{
  "api_version": "web/v1"
}
```

```
    }
```

metadata

description	Top-level scope that contains the web UI configuration's `name` and `created_by` information.
required	true
type	Map of key-value pairs

example	**YML**

```yaml
metadata:
  name: custom-web-ui
  created_by: admin
```

JSON

```json
{
  "metadata": {
    "name": "custom-web-ui",
    "created_by": "admin"
  }
}
```

spec

description	Top-level map that includes web UI configuration spec attributes.
required	Required for web UI configuration in `wrapped-json` or `yaml` format.
type	Map of key-value pairs

example	**YML**

```yaml
spec:
```

```
    always_show_local_cluster: false
    default_preferences:
      page_size: 50
      theme: classic
    link_policy:
      allow_list: true
      urls:
      - https://example.com
      - steamapp://34234234
      - "//google.com"
      - "//*.google.com"
      - "//bob.local"
      - https://grafana-
host/render/metrics?width=500&height=250#sensu.io.graphic
```

JSON

```
{
  "spec": {
    "always_show_local_cluster": false,
    "default_preferences": {
      "page_size": 50,
      "theme": "classic"
    },
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
      ]
    }
  }
}
```

Metadata attributes

name	
description	Name for the web UI configuration that is used internally by Sensu.
required	true
type	String
example	**YML**

```yml
name: custom-web-ui
```

JSON

```json
{
  "name": "custom-web-ui"
}
```

created_by	
description	Username of the Sensu user who created or last updated the web UI configuration. Sensu automatically populates the `created_by` field when the web UI configuration is created or updated. The admin user, cluster admins, and any user with access to the `GlobalConfig` resource can create and update web UI configurations.
required	false
type	String
example	**YML**

```yml
created_by: admin
```

JSON

```
{
  "created_by": "admin"
}
```

Spec attributes

always_show_local_cluster	
description	Use only in federated environments. Set to `true` to display the cluster the user is currently connected to in the <u>namespace switcher</u>. To omit local cluster details, set to `false` .
required	false
type	Boolean
default	`false`

example	**YML**
	
	```always_show_local_cluster: false```
	
	JSON
	
	```{\n "always_show_local_cluster": false\n}```

default_preferences	
description	Global default page size and theme preferences for all users.
required	false
type	Map of key-value pairs

example	
	YML

```yml
default_preferences:
  page_size: 50
  theme: classic
```

JSON

```json
{
  "default_preferences": {
    "page_size": 50,
    "theme": "classic"
  }
}
```

link_policy

description	For labels or annotations that contain a URL, the policy for which domains are valid and invalid targets for conversion to a link or an image.
required	false
type	Map of key-value pairs
	YML
example	

```yml
link_policy:
  allow_list: true
  urls:
  - https://example.com
  - steamapp://34234234
  - "//google.com"
  - "//*.google.com"
  - "//bob.local"
  - https://grafana-
host/render/metrics?width=500&height=250#sensu.io.graphic
```

JSON

```
{
  "link_policy": {
    "allow_list": true,
    "urls": [
      "https://example.com",
      "steamapp://34234234",
      "//google.com",
      "//*.google.com",
      "//bob.local",
      "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
    ]
  }
}
```

Default preferences attributes

page_size	
description	The number of items to list on each page.
required	false
type	Integer
default	25
example	**YML**
	``` page_size: 25 ```
	JSON

	``
	{
	  "page_size": 25
	}
	``

theme

description	The theme used in the web UI.

> **NOTE**: If an individual user's settings conflict with the web UI configuration settings, Sensu will use the individual user's settings. For example, if a user's system is set to dark mode and their web UI settings are configured to use their system settings, the web UI will use dark mode for that user, even if you set the theme to `classic` in your web UI configuration.

required	false
type	String
default	`sensu`
allowed values	`sensu`, `classic`, `uchiwa`, `tritanopia`, and `deuteranopia`
example	**YML**

```yml
theme: classic
```

JSON

```json
{
  "theme": "classic"
}
```

Link policy attributes

allow_list

description	If the list of URLs acts as an allow list, `true`. If the list of URLs acts as a deny list, `false`. As an allow list, only matching URLs will be

expanded. As a deny list, matching URLs will not be expanded, but any other URLs will be expanded.

required	false
type	Boolean
default	`false`

example	**YML**
	
	``yml
	allow_list: true
	``
	
	JSON
	
	``json
	{
	  "allow_list": true
	}
	``

urls

description	The list of URLs to use as an allow or deny list. *NOTE*: For images from services that may not have an easily distinguishable file extension, append the anchor `#sensu.io.graphic` to the image URLs.
required	false
type	Array

example	**YML**
	
	``yml
	urls:
	- https://example.com
	- steamapp://34234234
	- "//google.com"
	- "//*.google.com"
	- "//bob.local"
	``

```
  - https://grafana-
  host/render/metrics?width=500&height=250#sensu.io.graphic
```

JSON

```json
{
  "urls": [
    "https://example.com",
    "steamapp://34234234",
    "//google.com",
    "//*.google.com",
    "//bob.local",
    "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
  ]
}
```

API

API version: v2

The Sensu backend REST API provides a centrally managed control plane for automated, repeatable observability workflow configuration and observation data (event) access.

If you have a healthy clustered backend, you only need to make Sensu API calls to any one of the cluster members. The cluster protocol will replicate your changes to all cluster members.

For information about the Sensu agent API, read the agent reference.

Available APIs

Access all of the data and functionality of Sensu's first-class API clients, sensuctl and the web UI, with Sensu's backend REST APIs. Use the Sensu APIs to customize your workflows and integrate your favorite Sensu features with other tools and products.

- APIKeys API

- Assets API

- Authentication API

- Authentication providers API

- Checks API

- Cluster API

- Cluster role bindings API

- Cluster roles API

- Datastore API

- Entities API

- Events API

- Federation API

- Filters API

- ¬ Handlers API

- ¬ Health API

- ¬ Hooks API

- ¬ License API

- ¬ Metrics API

- ¬ Mutators API

- ¬ Namespaces API

- ¬ Prune API

- ¬ Role bindings API

- ¬ Roles API

- ¬ Searches API

- ¬ Secrets API

- ¬ Silencing API

- ¬ Tessen API

- ¬ Users API

- ¬ Version API

- ¬ Web UI configuration API

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. Read the RBAC reference for more information about configuring Sensu users and access controls.

Data format

The Sensu API uses JSON-formatted requests and responses.

In terms of output formats, the <u>Sensu API</u> uses `json` output format for responses for APIs in the `core` group. For APIs that are not in the `core` group, responses are in the `wrapped-json` output format. The `wrapped-json` format includes an outer-level `spec` "wrapping" for resource attributes and lists the resource `type` and `api_version`.

Sensu sends events to the backend in `json` <u>format</u>, without the `spec` attribute wrapper or `type` and `api_version` attributes.

Versioning

The Sensu Go API is versioned according to the format `v{majorVersion}{stabilityLevel}{iterationNumber}`, in which `v2` is stable version 2. The Sensu API guarantees backward compatibility for stable versions of the API.

Sensu does not guarantee that an alpha or beta API will be maintained for any period of time. Consider alpha versions under active development — they may not be published for every release. Beta APIs are more stable than alpha versions, but they offer similarly short-lived lifespans and also are not guaranteed to convert programmatically when the API is updated.

Request size limit

The default limit for API request body size is 0.512 MB. Use the `--api-request-limit` <u>backend configuration flag</u> to customize the API request body size limit if needed.

Access control

With the exception of the <u>authentication</u>, <u>health</u>, and <u>metrics</u> APIs, the Sensu API requires

authentication using a JSON Web Token (JWT) access token or API key.

Code examples in the Sensu API docs use the environment variable `$SENSU_API_KEY` to represent a valid API key in API requests.

> **NOTE**: *The authentication information on this page is specific to the Sensu API. For information about using Sensu's built-in basic authentication or external authentication providers to authenticate to the Sensu web UI, API, or sensuctl, read the Control Access documentation.*

Authentication quickstart

To set up a local API testing environment, save your Sensu credentials and access token as environment variables.

Save your Sensu credentials as environemnt variables:

```
export SENSU_USER=YOUR_USERNAME && SENSU_PASS=YOUR_PASSWORD
```

Save your Sensu access token as an environment variable:

> **NOTE**: *The command to save your access token as an environment variable requires curl and jq.*

```
export SENSU_ACCESS_TOKEN=`curl -X GET -u "$SENSU_USER:$SENSU_PASS" -s
http://localhost:8080/auth | jq -r ".access_token"`
```

The sensuctl reference demonstrates how to use the `sensuctl env` command to export your access token, token expiry time, and refresh token as environment variables.

Authenticate with the authentication API

Use the authentication API and your Sensu username and password to generate access tokens and refresh tokens. The `/auth` API endpoint lets you generate short-lived API tokens using your Sensu username and password.

1. Retrieve an access token for your user. For example, to generate an access token using example admin credentials:

```
curl -u 'YOUR_USERNAME:YOUR_PASSWORD' http://localhost:8080/auth
```

The access token should be included in the output, along with a refresh token:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1544582187,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
```

The access and refresh tokens are JWTs that Sensu uses to digitally sign the details of users' authenticated Sensu sessions.

2. Use the access token in the authentication header of the API request. For example:

```
curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events
```

3. Refresh your access token every 15 minutes. Access tokens last for approximately 15 minutes. When your token expires, you should receive a `401 Unauthorized` response from the API. To generate a new access token, use the `/auth/token` API endpoint, including the expired access token in the authorization header and the refresh token in the request body:

```
curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
-H 'Content-Type: application/json' \
-d '{"refresh_token": "eyJhbGciOiJIUzI1NiIs..."}' \
http://127.0.0.1:8080/auth/token
```

The new access token should be included in the output:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
```

```
    "expires_at": 1561055277,
    "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
  }
```

Generate an API access token with sensuctl

You can also generate an API access token using the sensuctl command line tool. The user credentials that you use to configure sensuctl determine your permissions to get, list, create, update, and delete resources with the Sensu API.

1. Install and configure sensuctl.

2. Retrieve an access token for your user:

   ```
   cat ~/.config/sensu/sensuctl/cluster|grep access_token
   ```

   The access token should be included in the output:

   ```
   "access_token": "eyJhbGciOiJIUzI1NiIs...",
   ```

3. Copy the access token into the authentication header of the API request. For example:

   ```
   curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
   http://127.0.0.1:8080/api/core/v2/namespaces/default/events
   ```

4. Refresh your access token every 15 minutes. Access tokens last for approximately 15 minutes. When your token expires, you should receive a `401 Unauthorized` response from the API. To regenerate a valid access token, run any sensuctl command (like `sensuctl event list`) and repeat step 2.

Authenticate with an API key

Each Sensu API key (core/v2.APIKey) is a persistent universally unique identifier (UUID) that maps to a stored Sensu username. The advantages of authenticating with API keys rather than access tokens

include:

- **More efficient integration**: Check and handler plugins and other code can integrate with the Sensu API without implementing the logic required to authenticate via the `/auth` API endpoint to periodically refresh the access token
- **Improved security**: API keys do not require providing a username and password in check or handler definitions
- **Better admin control**: API keys can be created and revoked without changing the underlying user's password, but keep in mind that API keys will continue to work even if the user's password changes

API keys are cluster-wide resources, so only cluster admins can grant, view, and revoke them.

> **NOTE**: API keys are not supported for authentication providers such as LDAP and OIDC.

Configure an environment variable for API key authentication

Configure the `SENSU_API_KEY` environment variable with your own API key to use it for authentication in your Sensu API requests as shown in the Sensu API code examples.

Follow these steps to generate an API key and export it to the `SENSU_API_KEY` environment variable:

1. Generate an API key with sensuctl:

   ```
   sensuctl api-key grant admin
   ```

   The response will include the new API key:

   ```
   Created: /api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
   ```

   > **PRO TIP**: Sensuctl is the most direct way to generate an API key, but you can also use the APIkeys API.

2. Export your API key to the `SENSU_API_KEY` environment variable:

```bash
export SENSU_API_KEY="83abef1e-e7d7-4beb-91fc-79ad90084d5b"
```

```cmd
SET SENSU_API_KEY="83abef1e-e7d7-4beb-91fc-79ad90084d5b"
```

```powershell
$Env:SENSU_API_KEY = "83abef1e-e7d7-4beb-91fc-79ad90084d5b"
```

Authorization header for API key authentication

Similar to the `Bearer [token]` Authorization header, `Key [api-key]` will be accepted as an Authorization header for authentication.

For example, a JWT `Bearer [token]` Authorization header might be:

```
curl -H "Authorization: Bearer $SENSU_ACCESS_TOKEN"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

If you're using `Key [api-key]` to authenticate instead, the Authorization header might be:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

Example

This example uses the API key directly (rather than the `$SENSU_API_KEY` environment variable) to authenticate to the checks API:

```
curl -H "Authorization: Key 7f63b5bc-41f4-4b3e-b59b-5431afd7e6a2"
```

```
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK` and the definitions for the checks in the default namespace.

Pagination

The Sensu API supports response pagination for most `core/v2` GET endpoints that return an array. You can request a paginated response with the `limit` and `continue` query parameters.

Limit query parameter

The following request limits the response to a maximum of two objects:

```
curl http://127.0.0.1:8080/api/core/v2/users?limit=2 -H "Authorization: Key
$SENSU_API_KEY"
```

The response includes the available objects up to the specified limit.

Continue query parameter

If more objects are available beyond the limit you specified in a request, the response header includes a `Sensu-Continue` token you can use to request the next page of objects.

For example, the following response indicates that more than two users are available because it provides a `Sensu-Continue` token in the response header:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Continue: L2R1ZmF1bU2Vuc3UtTWFjQ
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: Key $SENSU_API_KEY"
```

If the response header does not include a `Sensu-Continue` token, there are no further objects to return. For example, this response header indicates that no further users are available:

```
HTTP/1.1 200 OK
Content-Type: application/json
Sensu-Entity-Count: 3
Sensu-Entity-Limit: 100
Sensu-Entity-Warning:
Date: Fri, 14 Feb 2020 15:46:02 GMT
Content-Length: 54
[
  {
    "username": "alice",
    "groups": [
```

```
        "ops"
    ],
    "disabled": false
  }
]
```

Etag response headers

All GET and PATCH requests return an Etag HTTP response header that identifies a specific version of the resource.Use the Etag value from the response header to conditionally execute PATCH requests that use the If-Match and If-None-Match headers.

If Sensu cannot execute a PATCH request because one of the conditions failed, the request will return the HTTP response code `412 Precondition Failed`.

If-Match example

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-H 'If-Match: "drrn157624731797"' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK`.

If-None-Match example

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
```

```
  -H 'Content-Type: application/merge-patch+json' \
  -H 'If-None-Match: "drrn157624731797", "reew237527931897"' \
  -d '{
    "metadata": {
      "labels": {
        "region": "us-west-1"
      }
    }
  }' \
  http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
```

A successful request will return the HTTP response code `HTTP/1.1 200 OK`.

Response filtering

> **COMMERCIAL FEATURE**: Access API response filtering in the packaged Sensu Go distribution. For more information, read _Get started with commercial features_.

The Sensu API supports response filtering for all GET endpoints that return an array. You can filter resources based on their labels with the `labelSelector` query parameter and based on certain pre-determined fields with the `fieldSelector` query parameter.

> **NOTE**: To search based on fields and labels in the Sensu web UI, read _Search in the web UI_.

Label selector

The `labelSelector` query parameter allows you to group resources by the label attributes specified in the resource metadata object. All resources support labels within the metadata object.

Field selector

The `fieldSelector` query parameter allows you to organize and select subsets of resources based on certain fields. Here's the list of available fields:

Resource	Fields
Asset	`asset.name` `asset.namespace` `asset.filters`
Check	`check.name` `check.namespace` `check.handlers` `check.publish` `check.round_robin` `check.runtime_assets` `check.subscriptions`
ClusterRole	`clusterrole.name`
ClusterRoleBinding	`clusterrolebinding.name` `clusterrolebinding.role_ref.name` `clusterrolebinding.role_ref.type`
Entity	`entity.name` `entity.namespace` `entity.deregister` `entity.entity_class` `entity.subscriptions`
Event	`event.is_silenced` `event.name` `event.namespace` `event.check.handlers` `event.check.is_silenced` `event.check.name` `event.check.publish` `event.check.round_robin` `event.check.runtime_assets` `event.check.status` `event.check.subscriptions` `event.entity.deregister` `event.entity.entity_class` `event.entity.name` `event.entity.subscriptions`
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`
Secrets	`secret.name` `secret.namespace` `secret.provider` `secret.id`
SecretsProviders	`provider.name` `provider.namespace`

Silenced	`silenced.name` `silenced.namespace` `silenced.check`
	`silenced.creator` `silenced.expire_on_resolve`
	`silenced.subscription`
User	`user.username` `user.disabled` `user.groups`

API-specific syntax

To create an API response filter, you'll write a brief filter statement. The <u>operators</u> and <u>examples</u> sections demonstrate how to construct API response filter statements for different operators and specific purposes.

The filter statement construction is slightly different for different operators, but there are a few general syntax rules that apply to all filter statements.

Spaces in the filter statement

As shown in this example:

```
'fieldSelector=silenced.expire_on_resolve == true'
```

- ⌐ **Do not** use spaces around the `=` between the selector type and the rest of the filter statement.
- ⌐ **Do** use spaces around the operator (in this example, the `==`).

Quotation marks around the filter statement

Place the entire filter statement inside single quotes:

```
'fieldSelector=linux in check.subscriptions'
```

Exception: If the filter statement contains a *shell* variable, you must use double quotation marks around the statement:

```
"labelSelector=host == $HOSTNAME"
```

If you use single quotes around a filter statement that contains a shell variable, the single quotes will keep the variable intact instead of expanding it.

> **NOTE**: This exception only applies to shell variables. It does not apply for variables in languages that treat single and double quotation marks interchangeably, like JavaScript.

Values that begin with a number or include special characters

If you are filtering for a value that begins with a number, place the value in double quotes:

```
'fieldSelector=entity.name == "1b04994n"'
```

Likewise, to use a label or field selector with string values that include special characters like hyphens and underscores, place the value in double quotes:

```
'labelSelector:region == "us-west-1"'
```

Operators

Sensu's API response filtering supports two equality-based operators, two set-based operators, one substring matching operator, and one logical operator.

operator	description	example
==	Equality	`check.publish == true`
!=	Inequality	`check.namespace != "default"`
in	Included in	`linux in check.subscriptions`
notin	Not included in	`slack notin check.handlers`
```
```

matches	Substring matching	`check.name matches "linux-"`
`&&`	Logical AND	`check.publish == true && slack in` `check.handlers`

Equality-based operators

Sensu's two *equality-based* operators are `==` (equality) and `!=` (inequality).

For example, to retrieve only checks with the label `type` and value `server`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'labelSelector=type == "server"'
```

> **NOTE**: Use the flag `--data-urlencode` in cURL to encode the query parameter. Include the `-G` flag so the request appends the query parameter data to the URL.

To retrieve checks that are not in the `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.namespace != "production"'
```

Set-based operators

Sensu's two *set-based* operators for lists of values are `in` and `notin`.

For example, to retrieve checks with a `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=linux in check.subscriptions'
```

To retrieve checks that do not use the `slack` handler:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=slack notin check.handlers'
```

The `in` and `notin` operators have two important conditions:

- ¬ First, they only work when the underlying value you're filtering for is a string. You can filter for strings and arrays of strings with `in` and `notin` operators, but you cannot use them to filter for integer, float, array, or Boolean values.

- ¬ Second, to filter for a string, the string must be to the **left** of the operator: `string [in|notin] selector` . To filter for an array of strings, the array must be to the **right** of the operator: `selector [in|notin] [string1,string2]` .

Substring matching operator

Sensu's *substring matching* operator is `matches` .

For example, to retrieve all checks whose name includes `linux` :

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.name matches "linux"'
```

Suppose you are using Sensu to monitor 1000 entities that are named incrementally and according to technology. For example, your webservers are named `webserver-1` through `webserver-25` , and your CPU entities are named `cpu-1` through `cpu-300` , and so on. In this case, you can use `matches` to retrieve all of your `webserver` entities:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector=entity.name matches "webserver-"'
```

Similarly, if you have entities labeled for different regions, you can use `matches` to find the entities that are labeled for the US (for example, `us-east-1`, `us-west-1`, and so on):

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'labelSelector:region matches "us"'
```

The `matches` operator only works when the underlying value you're filtering for is a string. You can filter for strings and arrays of strings with the `matches` operator, but you cannot use it to filter for integer, float, array, or Boolean values. Also, the string must be to the **right** of the operator: `selector matches string`.

Logical operator

Sensu's logical operator is `&&` (AND). Use it to combine multiple statements separated with the logical operator in field and label selectors.

For example, the following cURL request retrieves checks that are not configured to be published **and** include the `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.publish != true && linux in
check.subscriptions'
```

To retrieve checks that are not published, include a `linux` subscription, and are in the `dev` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=check.publish != true && linux in check.subscriptions
&& dev in check.namespace'
```

NOTE: Sensu does not have the `OR` logical operator.

Combined selectors

You can use field and label selectors in a single request. For example, to retrieve only checks that include a `linux` subscription *and* do not include a label for type `server`:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks -G \
--data-urlencode 'fieldSelector=linux in check.subscriptions' \
--data-urlencode 'labelSelector=type != "server"'
```

Examples

Values with special characters

To use a label or field selector with string values that include special characters like hyphens and underscores, place the value in single or double quotes:

```
curl -H "Authorization: Key $SENSU_API_KEY" -X GET
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'labelSelector=region == "us-west-1"'
```

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector="entity:i-0c1f8a116b84ea50c" in entity.subscriptions'
```

Use selectors with arrays of strings

To retrieve checks that are in either the `dev` or `production` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks -G \
--data-urlencode 'fieldSelector=check.namespace in [dev,production]'
```

Filter events by entity or check

To retrieve events for a specific check (`checkhttp`):

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=checkhttp in event.check.name'
```

Similary, to retrieve only events for the `server` entity:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=server in event.entity.name'
```

Filter events by severity

Use the `event.check.status` field selector to retrieve events by severity. For example, to retrieve all events at `2` (CRITICAL) status:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=event.check.status == "2"'
```

Filter all incidents

To retrieve all incidents (all events whose status is not `0`):

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=event.entity.status != "0"'
```

Filter checks, entities, or events by subscription

To list all checks that include the `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/checks
-G \
--data-urlencode 'fieldSelector=linux in check.subscriptions'
```

Similarly, to list all entities that include the `linux` subscription:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/entities -G \
--data-urlencode 'fieldSelector=linux in entity.subscriptions'
```

To list all events for the `linux` subscription, use the `event.entity.subscriptions` field selector:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=linux in event.entity.subscriptions'
```

Filter silenced resources and silences

Filter silenced resources by namespace

To list all silenced resources for a particular namespace (in this example, the `default` namespace):

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.namespace == "default"'
```

Likewise, to list all silenced resources *except* those in the `default` namespace:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.namespace != "default"'
```

To list all silenced events for all namespaces:

```
curl -H "Authorization: Key $SENSU_API_KEY" http://127.0.0.1:8080/api/core/v2/events
-G \
--data-urlencode 'fieldSelector=event.is_silenced == true'
```

Filter silences by creator

To list all silences created by the user `alice`:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.creator == "alice"'
```

To list all silences that were not created by the `admin` user:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.creator != "admin"'
```

Filter silences by silence subscription

To retrieve silences with a specific subscription (in this example, `linux`):

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.subscription == "linux"'
```

Another way to make the same request is:
```

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=linux in silenced.subscription'
```

> **NOTE**: For this field selector, `subscription` means the subscription specified for the silence. In other words, this filter retrieves **silences** with a particular subscription, not silenced entities or checks with a matching subscription.

Filter silenced resources by expiration

To list all silenced resources that expire only when a matching check resolves:

```
curl -H "Authorization: Key $SENSU_API_KEY"
http://127.0.0.1:8080/api/core/v2/silenced -G \
--data-urlencode 'fieldSelector=silenced.expire_on_resolve == true'
```

APIKeys API

Get all API keys

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK

[
  {
    "metadata": {
      "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b",
      "created_by": "admin"
    },
    "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. Read the API overview for details.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output	<code>[
	  {
	    "metadata": {
	      "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b",
	      "created_by": "admin"
	    },
	    "username": "admin",
	    "created_at": 1570640363
	  }
	]</code>

Create a new API key

The `/apikeys` API endpoint provides HTTP POST access to create a new API key.

Example

In the following example, an HTTP POST request is submitted to the `/apikeys` API endpoint to

create a new API key. The request returns a successful HTTP `201 Created` response, along with a `Location` header that contains the relative path to the new API key.

> **NOTE**: For the `/apikeys` POST endpoint, authenticate with a Sensu access token, which you can generate with the [authentication API](#) or [sensuctl](#). This example uses `$SENSU_ACCESS_TOKEN` to represent a valid Sensu access token.
>
> If you prefer, you can [create a new API key with sensuctl](#) instead of using this endpoint.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_ACCESS_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "username": "admin"
}' \
http://127.0.0.1:8080/api/core/v2/apikeys


HTTP/1.1 201 Created
```

API Specification

/apikeys (POST)	
description	Creates a new API key, a Sensu-generated universally unique identifier (UUID). The response will include HTTP 201 and a `Location` header that contains the relative path to the new API key.
example URL	http://hostname:8080/api/core/v2/apikeys
request payload	```{ "username": "admin" }```
response codes	˥ **Success**: 201 (Created)
	˥

- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific API key

The `/apikeys/:apikey` GET endpoint retrieves the specified API key.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b",
    "created_by": "admin"
  },
  "username": "admin",
  "created_at": 1570640363
}
```

API Specification

/apikeys/:apikey (GET)	
description	Returns the specified API key.
example url	http://hostname:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b

response type	Map

response codes	
	⌐ **Success**: 200 (OK)
	⌐ **Missing**: 404 (Not Found)
	⌐ **Error**: 500 (Internal Server Error)

output	
| | ```json
{
  "metadata": {
    "name": "83abef1e-e7d7-4beb-91fc-79ad90084d5b",
    "created_by": "admin"
  },
  "username": "admin",
  "created_at": 1570640363
}
``` |

Update an API key with PATCH

The `/apikeys/:apikey` PATCH endpoint updates the specified API key.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.

Example

In the following example, querying the `/apikeys/:apikey` API updates the username for the specified `:apikey` definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
```

```
{
  "username": "devteam"
} \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b


HTTP/1.1 200 OK
```

API Specification

/apikeys/:apikey (PATCH)	
description	Updates the specified API key.
example url	http://hostname:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	``` { "username": "devteam" } ```

Delete an API key

The `/apikeys/:apikey` API endpoint provides HTTP DELETE access to remove an API key.

Example

The following example shows a request to the `/apikeys/:apikey` API endpoint to delete the API key `83abef1e-e7d7-4beb-91fc-79ad90084d5b`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/apikeys/83abef1e-e7d7-4beb-91fc-79ad90084d5b

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

Get all assets

The `/assets` API endpoint provides HTTP GET access to dynamic runtime asset data.

Example

The following example demonstrates a request to the `/assets` API endpoint, resulting in a JSON array that contains dynamic runtime asset definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets \
-H "Authorization: Key $SENSU_API_KEY"

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
1257bb479676bc155b24e21bf93c722b812b0f15cb3bd",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-influxdb-handler",
```

```
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  },
  {
    "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
    "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "builds": null,
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

API Specification

/assets (GET)

description	Returns the list of dynamic runtime assets.
example url	http://hostname:8080/api/core/v2/namespaces/default/assets
pagination	This endpoint supports pagination using the `limit` and `continue`

	query parameters.
response filtering	This endpoint supports <u>API response filtering</u>.
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",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  },
  {
    "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-
handler_1.0.3_linux_amd64.tar.gz",
    "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66
556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eedb706
```

```
        4df9019adc8",
            "filters": [
                "entity.system.os == 'linux'",
                "entity.system.arch == 'amd64'"
            ],
            "builds": null,
            "metadata": {
                "name": "sensu-slack-handler",
                "namespace": "default",
                "created_by": "admin"
            },
            "headers": {
                "Authorization": "Bearer $TOKEN",
                "X-Forwarded-For": "client1, proxy1, proxy2"
            }
        }
    ]
```

Create a new dynamic runtime asset

The `/assets` API endpoint provides HTTP POST access to dynamic runtime asset data.

Example

In the following example, an HTTP POST request is submitted to the `/assets` API endpoint to create a role named `sensu-slack-handler`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
  "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
```

```
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
  "filters": [
    "entity.system.os == 'linux'",
    "entity.system.arch == 'amd64'"
  ],
  "headers": {
    "Authorization": "Bearer $TOKEN",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  },
  "metadata": {
    "name": "sensu-slack-handler",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets


HTTP/1.1 201 Created
```

API Specification

/assets (POST)	
description	Creates a Sensu dynamic runtime asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets

payload	<pre>{
	  "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": {</pre>

```
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    },
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default"
    }
  }
```

---

response codes

- ⌐ **Success**: 201 (Created)

- ⌐ **Malformed**: 400 (Bad Request)

- ⌐ **Error**: 500 (Internal Server Error)

Get a specific dynamic runtime asset

The `/assets/:asset` API endpoint provides HTTP GET access to <u>dynamic runtime asset data</u> 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 <u>:asset</u> <u>definition</u> (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: Key $SENSU_API_KEY"

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",
      "created_by": "admin"
    },
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
]
```

API Specification

/assets/:asset (GET)	
description	Returns the specified dynamic runtime asset.
example url	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output	<pre>[
	  {
	    "url": "https://github.com/sensu/sensu-slack-handler/releases/download/1.0.3/sensu-slack-</pre>

```
        handler_1.0.3_linux_amd64.tar.gz",
            "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a
66556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eed
b7064df9019adc8",
            "filters": [
                "entity.system.os = 'linux'",
                "entity.system.arch = 'amd64'"
            ],
            "builds": null,
            "metadata": {
                "name": "sensu-slack-handler",
                "namespace": "default",
                "created_by": "admin"
            },
            "headers": {
                "Authorization": "Bearer $TOKEN",
                "X-Forwarded-For": "client1, proxy1, proxy2"
            }
        }
    ]
```

Create or update a dynamic runtime asset

The `/assets/:asset` API endpoint provides HTTP PUT access to create or update specific `:asset` definitions, by dynamic runtime asset name.

Example

In the following example, an HTTP PUT request is submitted to the `/assets/:asset` API endpoint to create the dynamic runtime asset `sensu-slack-handler`. The request returns a successful HTTP `20 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
```

```
    "url": "https://github.com/sensu/sensu-slack-
handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
    "sha512":
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b58714
73e6c851f51b34097c54fdb8d18eedb7064df9019adc8",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    },
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default"
    }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/sensu-slack-
handler


HTTP/1.1 201 Created
```

API Specification

/assets/:asset (PUT)	
description	Creates or updates the specified Sensu dynamic runtime asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler

payload	```{
	  "url": "https://github.com/sensu/sensu-slack-handler/releases/download/1.0.3/sensu-slack-handler_1.0.3_linux_amd64.tar.gz",
	  "sha512": "68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b5871473e6c851f51b34097c54fdb8d18eed```

```json
    b7064df9019adc8",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "headers": {
      "Authorization": "Bearer $TOKEN",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    },
    "metadata": {
      "name": "sensu-slack-handler",
      "namespace": "default"
    }
  }
}
```

---

response codes

¬ **Success**: 201 (Created)

¬ **Malformed**: 400 (Bad Request)

¬ **Error**: 500 (Internal Server Error)

Update a dynamic runtime asset with PATCH

The `/assets/:asset` API endpoint provides HTTP PATCH access to update `:asset` definitions, specified by asset name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/assets/:asset` API endpoint to add a label for the `sensu-slack-handler` asset, resulting in an HTTP `200 OK` response and the

updated asset definition.

We support [JSON merge patches][4], so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler

HTTP/1.1 200 OK
```

API Specification

/assets/:asset (PATCH)	
description	Updates the specified Sensu asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler

payload	<pre>{
	  "metadata": {
	    "labels": {
	      "region": "us-west-1"
	    }
	  }
	}</pre>

response codes	

- ¬ **Success**: 200 (OK)
- ¬ **Malformed**: 400 (Bad Request)
- ¬ **Error**: 500 (Internal Server Error)

Delete a dynamic runtime asset

The `/assets/:asset` API endpoint provides HTTP DELETE access so you can delete a dynamic runtime assets.

> **NOTE**: *Deleting a dynamic runtime asset does not remove the downloaded files from the asset cache or remove any references to the deleted asset in other resources.*

Example

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 204 No Content
```

API Specification

/assets/:asset (DELETE)	
description	Deletes the specified Sensu dynamic runtime asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/sensu-slack-handler
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬

¬ **Error**: 500 (Internal Server Error)

Authentication API

Generate an access token and a refresh token

The `/auth` API endpoint provides HTTP GET access to generate an access token and a refresh token using Sensu's basic authentication.

The access and refresh tokens are JSON Web Tokens (JWTs) that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature.

Example

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 receive a `401 Unauthorized` response from the API. To generate a new access token, use the `/auth/token` API endpoint.
example url	http://hostname:8080/auth
| output | ```json
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "expires_at": 1544582187,
  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
}
``` |
| response codes | ¬ **Valid credentials**: 200 (OK)

¬ **Invalid credentials**: 401 (Unauthorized)

¬ **Error**: 500 (Internal Server Error) |

Test basic auth user credentials

The `/auth/test` API endpoint provides HTTP GET access to test basic authentication user credentials that were created with Sensu's built-in basic authentication.

> NOTE: The `/auth/test` endpoint only tests user credentials created with Sensu's built-in basic authentication provider. It does not test user credentials defined via an authentication provider like Lightweight Directory Access Protocol (LDAP), Active Directory (AD), or OpenID Connect 1.0 protocol (OIDC).

Example

In the following example, querying the `/auth/test` API endpoint with a given username and password 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	http://hostname:8080/auth/test
response codes	**Valid credentials**: 200 (OK) **Invalid credentials**: 401 (Unauthorized) **Error**: 500 (Internal Server Error)

Renew an access token

The `/auth/token` API endpoint provides HTTP POST access to renew an access token.

Example

In the following example, an HTTP POST request is submitted to the `/auth/token` API endpoint to generate a valid access token. The request includes the refresh token in the request body and returns a successful HTTP `200 OK` response along with the new access token.

The access and refresh tokens are <u>JSON Web Tokens (JWTs)</u> that Sensu issues to record the details of users' authenticated Sensu sessions. The backend digitally signs these tokens, and the tokens can't be changed without invalidating the signature.

```
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	<pre>{
	  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
	}</pre>

output	<pre>{
	  "access_token": "eyJhbGciOiJIUzI1NiIs...",
	  "expires_at": 1544582187,
	  "refresh_token": "eyJhbGciOiJIUzI1NiIs..."
	}</pre>

response codes	⌐ **Success**: 200 (OK) ⌐ **Malformed**: 400 (Bad Request)

¬ **Error**: 500 (Internal Server Error)

Authentication providers API

NOTE: *Requests to the authentication providers API require you to authenticate with a Sensu* <u>*API key*</u> *or* <u>*access token*</u>. *The code examples in this document use the* <u>*environment variable*</u> `$SENSU_API_KEY` *to represent a valid API key in API requests.*

Get active authentication provider configurations

The `/authproviders` API endpoint provides HTTP GET access to authentication provider configuration in Sensu.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "type": "ldap",
    "api_version": "authentication/v2",
    "metadata": {
      "name": "openldap"
    },
    "spec": {
      "groups_prefix": "",
```

```json
      "servers": [
        {
          "binding": {
            "password": "YOUR_PASSWORD",
            "user_dn": "cn=binder,dc=acme,dc=org"
          },
          "client_cert_file": "",
          "client_key_file": "",
          "default_upn_domain": "",
          "group_search": {
            "attribute": "member",
            "base_dn": "dc=acme,dc=org",
            "name_attribute": "cn",
            "object_class": "groupOfNames"
          },
          "host": "127.0.0.1",
          "insecure": false,
          "port": 636,
          "security": "tls",
          "trusted_ca_file": "",
          "user_search": {
            "attribute": "uid",
            "base_dn": "dc=acme,dc=org",
            "name_attribute": "cn",
            "object_class": "person"
          }
        }
      ],
      "username_prefix": ""
    }
  }
]
```

API Specification

/authproviders (GET)	
description	Returns the list of active authentication providers.
example url	http://hostname:8080/api/enterprise/authentication/v2/authproviders

query parameters	`types` : Defines which type of authentication provider to retrieve. Join with `&` to retrieve multiple types: `?types=AD&types=OIDC` .
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters. Read the API overview for details.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "type": "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": ""
        }
      }
    ]
```

Get the configuration for a specific authentication provider

The `/authproviders/:name` API endpoint provides HTTP GET access to the authentication provider configuration for a specific `:name`.

Example

In the following example, an HTTP GET request is submitted to the `/authproviders/:name` API endpoint to retrieve the `openldap` authentication 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: Key $SENSU_API_KEY" \
-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	
	
	``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": ""
  }
}
```

Create or update the configuration for a specific authentication provider

The `/authproviders/:name` API endpoint provides HTTP PUT access to create or update the authentication provider configuration for a specific `:name`.

Example

In the following example, an HTTP PUT request is submitted to the `/authproviders/:name` API endpoint to create the `openldap` authentication provider, resulting in an HTTP `200 OK` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "Type": "ldap",
  "api_version": "authentication/v2",
  "spec": {
    "servers": [
      {
        "host": "127.0.0.1",
        "binding": {
          "user_dn": "cn=binder,dc=acme,dc=org",
          "password": "YOUR_PASSWORD"
```

```
        },
        "group_search": {
          "base_dn": "dc=acme,dc=org"
        },
        "user_search": {
          "base_dn": "dc=acme,dc=org"
        }
      }
    ]
  },
  "metadata": {
    "name": "openldap"
  }
}' \
http://127.0.0.1:8080/api/enterprise/authentication/v2/authproviders/openldap


HTTP/1.1 200 OK
```

API Specification

/authproviders/:name (PUT)	
description	Creates or updates the authentication provider configuration for the specified name. Read Configure single sign-on (SSO) authentication for more information about supported providers.
example url	http://hostname:8080/api/enterprise/authentication/v2/authproviders/openldap

payload	<pre>{
	  "Type": "ldap",
	  "api_version": "authentication/v2",
	  "spec": {
	    "servers": [
	      {
	        "host": "127.0.0.1",
	        "binding": {
	          "user_dn": "cn=binder,dc=acme,dc=org",
	          "password": "YOUR_PASSWORD"</pre>

```
                },
                "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, read Configure single sign-on (SSO) authentication.
response codes	**Success**: 200 (OK) **Malformed**: 400 (Bad Request) **Error**: 500 (Internal Server Error)

Delete the configuration for a specific authentication provider

The `/authproviders/:name` API endpoint provides HTTP DELETE access to delete the authentication provider configuration from Sensu for a specific `:name`.

Example

The following example shows a request to the `/authproviders/:name` API endpoint to delete the configuration for the authentication provider `openldap`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/authproviders/openldap


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

Get all checks

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: Key $SENSU_API_KEY"

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"
    ],
```

```json
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "output_metric_tags": null,
      "env_vars": null,
      "metadata": {
        "name": "check-email",
        "namespace": "default",
        "created_by": "admin"
      }
    }
  ]
```

API Specification

/checks (GET)	
description	Returns the list of checks.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "command": "check-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,
    "output_metric_tags": null,
    "env_vars": null,
    "metadata": {
      "name": "check-email",
      "namespace": "default",
      "created_by": "admin"
    }
  }
]
```

Create a new check

The `/checks` API endpoint provides HTTP POST access to create checks.

Example

In the following example, an HTTP POST request is submitted to the `/checks` API endpoint to create a `check-cpu` check. The request includes the check definition in the request body and returns a successful HTTP `200 OK` response and the created check definition.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-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	`{`
	`  "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, read the checks reference.

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Get a specific check

The `/checks/:check` API endpoint provides HTTP GET access to check 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: Key $SENSU_API_KEY"
```

```
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,
  "output_metric_tags": null,
  "env_vars": null,
  "metadata": {
    "name": "check-cpu",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

API Specification

/checks/:check (GET)	
description	Returns the specified check.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check-

	cpu
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output

```
{
  "command": "check-cpu.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,
  "output_metric_tags": null,
  "env_vars": null,
  "metadata": {
    "name": "check-cpu",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

Create or update a check

The `/checks/:check` API endpoint provides HTTP PUT access to create and update `:check` definitions, specified by check name.

Example

In the following example, an HTTP PUT request is submitted to the `/checks/:check` API endpoint to update the `check-cpu` check, resulting in an HTTP `200 OK` response and the updated check definition.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-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 | ```json
{
  "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, read the checks reference. |
| response codes | ¬ **Success**: 201 (Created)

¬ **Malformed**: 400 (Bad Request)

¬ **Error**: 500 (Internal Server Error) |

Update a check with PATCH

The `/checks/:check` API endpoint provides HTTP PATCH access to update `:check` definitions,

specified by check name.

> **NOTE**: *You cannot change a resource's `name` or `namespace` with a PATCH request.Use a [PUT request](#) instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/checks/:check` API endpoint to update the subscriptions array for the `check-cpu` check, resulting in an HTTP `200 OK` response and the updated check definition.

We support [JSON merge patches](#), so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "subscriptions": [
    "linux",
    "health"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu

HTTP/1.1 200 OK
```

API Specification

/checks/:check (PATCH)	
description	Updates the specified Sensu check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/chec

payload

```json
{
  "subscriptions": [
    "linux",
    "health"
  ]
}
```

response codes

- ⌐ **Success**: 200 (OK)
- ⌐ **Malformed**: 400 (Bad Request)
- ⌐ **Error**: 500 (Internal Server Error)

Delete a check

The `/checks/:check` API endpoint provides HTTP DELETE access to delete a check from Sensu, specified by the check name.

Example

The following example shows a request to the `/checks/:check` API endpoint to delete the check named `check-cpu`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu

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)

Create an ad hoc check execution request

The `/checks/:check/execute` API endpoint provides HTTP POST access to create an ad hoc check execution request so you can execute a check on demand.

Example

In the following example, an HTTP POST request is submitted to the `/checks/:check/execute` API endpoint to execute the `check-cpu` check. The request includes the check name in the request body and returns a successful HTTP `202 Accepted` response and an `issued` timestamp.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "check": "check-cpu",
  "subscriptions": [
    "entity:i-424242"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu/execute

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-cpu/execute
payload	```json { "check": "check-cpu", "subscriptions": ["entity:i-424242"] } ```
payload parameters	¬ Required: `check` (the name of the check to execute). ¬ Optional: `subscriptions` (an array of subscriptions to publish the check request to). When provided with the request, the `subscriptions` attribute overrides any subscriptions configured in the check definition.
response codes	¬ **Success**: 202 (Accepted) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Assign a hook to a check

The `/checks/:check/hooks/:type` API endpoint provides HTTP PUT access to assign a hook to a check.

Example

In the following example, an HTTP PUT request is submitted to the `/checks/:check/hooks/:type` API endpoint, assigning the `process_tree` hook to the `check-cpu` check in the event of a `critical` type check result, resulting in a successful HTTP `204 No Content` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "critical": [
    "process_tree"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu/hooks/critical

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 <u>check response types</u> (for example, `critical` or `warning`). Each must contain an array of hook names.
response codes	⌐ **Success**: 201 (Created) ⌐ **Malformed**: 400 (Bad Request) ⌐ **Error**: 500 (Internal Server Error)

Remove a hook from a check

The `/checks/:check/hooks/:type/hook/:hook` API endpoint provides HTTP DELETE access to a remove a <u>hook</u> from a <u>check</u>.

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: Key $SENSU_API_KEY" \
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). Read the checks reference for available types.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check-cpu/hooks/critical/hook/process_tree
response codes	⌐ **Success**: 204 (No Content) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

Cluster API

Get all cluster data

The `/cluster/members` API endpoint provides HTTP GET access to Sensu cluster data.

Example

The following example demonstrates a request to the `/cluster/members` API endpoint, 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: Key $SENSU_API_KEY" \

HTTP/1.1 200 OK
{
  "header": {
    "cluster_id": 4255616304056076734,
    "member_id": 9882886658148554927,
    "raft_term": 2
  },
  "members": [
    {
      "ID": 9882886658148554927,
      "name": "default",
      "peerURLs": [
        "http://127.0.0.1:2380"
      ],
      "clientURLs": [
```

```
          "http://127.0.0.1:2379"
        ]
      }
    ]
  }
```

API Specification

/cluster/members (GET)

description	Returns the etcd cluster definition.
example url	http://hostname:8080/api/core/v2/cluster/members
query parameters	`timeout` : Defines the timeout when querying etcd. Default is `3` .
response type	Map
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)

example output	```
	{
	  "header": {
	    "cluster_id": 4255616304056076734,
	    "member_id": 9882886658148554927,
	    "raft_term": 2
	  },
	  "members": [
	    {
	      "ID": 9882886658148554927,
	      "name": "default",
	      "peerURLs": [
	        "http://127.0.0.1:2380"
	      ],
	      "clientURLs": [
	        "http://127.0.0.1:2379"
	      ]
	``

```
        }
    ]
}
```

Create a new cluster member

The `/cluster/members` API endpoint provides HTTP POST access to create a Sensu cluster member.

Example

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/cluster/members?peer-addrs=http://127.0.0.1:2380

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)

Create or update a cluster member

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: Key $SENSU_API_KEY" \
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)

Delete a cluster member

The `/cluster/members/:member` API endpoint provides HTTP DELETE access to remove a Sensu cluster member.

Example

The following example shows a request to the `/cluster/members/:member` API endpoint to remove the Sensu cluster member with the ID `8927110dc66458af`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
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)

Get a cluster ID

The `/cluster/id` API endpoint provides HTTP GET access to the Sensu cluster ID.

Example

The following example demonstrates a request to the `/cluster/id` API endpoint, resulting in a string that contains the Sensu cluster ID.

```
curl -X GET \
 -H "Authorization: Key $SENSU_API_KEY" \
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
query parameters	`timeout` : Defines the timeout when querying etcd. Default is `3` .
response type	String
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

example output

```
"23481e76-5844-4d07-b714-6e2ffbbf9315"
```

Cluster role bindings API

Get all cluster role bindings

The `/clusterrolebindings` API endpoint provides HTTP GET access to <u>cluster role binding</u> data.

Example

The following example demonstrates a request to the `/clusterrolebindings` API endpoint, resulting in a JSON array that contains <u>cluster role binding definitions</u>.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "cluster-admins"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-admin"
    },
    "metadata": {
      "name": "cluster-admin",
```

```
        "created_by": "admin"
      }
    },
    {
      "subjects": [
        {
          "type": "Group",
          "name": "system:agents"
        }
      ],
      "role_ref": {
        "type": "ClusterRole",
        "name": "system:agent"
      },
      "metadata": {
        "name": "system:agent",
        "created_by": "admin"
      }
    }
  ]
```

API Specification

/clusterrolebindings (GET)	
description	Returns the list of cluster role bindings.
example url	http://hostname:8080/api/core/v2/clusterrolebindings
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output

```
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "cluster-admins"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-admin"
    },
    "metadata": {
      "name": "cluster-admin",
      "created_by": "admin"
    }
  },
  {
    "subjects": [
      {
        "type": "Group",
        "name": "system:agents"
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "system:agent"
    },
    "metadata": {
      "name": "system:agent"
    }
  }
]
```

Create a new cluster role binding

The `/clusterrolebindings` API endpoint provides HTTP POST access to create a cluster role

binding.

Example

In the following example, an HTTP POST request is submitted to the `/clusterrolebindings` API endpoint to create a cluster role binding that assigns the `cluster-admin` cluster role to the user `bob` .The request includes the cluster role binding definition in the request body and returns a successful HTTP `200 OK` response and the created cluster role binding definition.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings

HTTP/1.1 201 Created
```

API Specification

/clusterrolebindings (POST)	
description	Creates a Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings

payload

```json
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}
```

response codes

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Get a specific cluster role binding

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP GET access to cluster role binding data for specific `:clusterrolebinding` definitions, by cluster role binding `name`.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder",
    "created_by": "admin"
  }
}
```

API Specification

/clusterrolebindings/:clusterrolebinding (GET)	
description	Returns the specified cluster role binding.
example url	http://hostname:8080/api/core/v2/clusterrolebindings/bob-binder
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	{

```json
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder",
    "created_by": "admin"
  }
}
```

Create or update a cluster role binding

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP PUT access to create or update a cluster role binding, by cluster role binding `name`.

Example

In the following example, an HTTP PUT request is submitted to the `/clusterrolebindings/:clusterrolebinding` API endpoint to create a cluster role binding that assigns the `cluster-admin` cluster role to users in the group `ops` . The request includes the cluster role binding definition in the request body and returns a successful HTTP `200 OK` response and the created cluster role binding definition.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "ops"
```

```
      }
    ],
    "role_ref": {
      "type": "ClusterRole",
      "name": "cluster-admin"
    },
    "metadata": {
      "name": "ops-group-binder"
    }
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-group-binder


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": {
	    "type": "ClusterRole",
	    "name": "cluster-admin"
	  },
	  "metadata": {
	    "name": "ops-group-binder"
	  }
	}</pre>

response codes

¬ **Success**: 201 (Created)

¬ **Malformed**: 400 (Bad Request)

¬ **Error**: 500 (Internal Server Error)

Update a cluster role binding with PATCH

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP PATCH access to update `:clusterrolebinding` definitions, specified by cluster role binding name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/clusterrolebindings/:clusterrolebinding` API endpoint to update the subjects array for the `ops-group-binder` cluster role binding, resulting in an HTTP `200 OK` response and the updated cluster role binding definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "subjects": [
    {
      "type": "Group",
```

```
      "name": "ops_team_1"
    },
    {
      "type": "Group",
      "name": "ops_team_2"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-group-binder

HTTP/1.1 200 OK
```

API Specification

/clusterrolebindings/:clusterrolebinding (PATCH)	
description	Updates the specified Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings/ops-group-binder

payload	<pre>{
	  "subjects": [
	    {
	      "type": "Group",
	      "name": "ops_team_1"
	    },
	    {
	      "type": "Group",
	      "name": "ops_team_2"
	    }
	  ]
	}</pre>

response codes	⌐ **Success**: 200 (OK) ⌐

Delete a cluster role binding

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP DELETE access to delete a cluster role binding from Sensu (specified by the cluster role binding name).

Example

The following example shows a request to the `/clusterrolebindings/:clusterrolebinding` API endpoint to delete the cluster role binding `ops-binding`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-binding

HTTP/1.1 204 No Content
```

API Specification

/clusterrolebindings/:clusterrolebinding (DELETE)	
description	Removes a cluster role binding from Sensu (specified by the cluster role binding name).
example url	http://hostname:8080/api/core/v2/clusterrolebindings/ops-binding
response codes	

- **Success**: 204 (No Content)

- ⌐

- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Cluster roles API

Get all cluster roles

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "assets",
          "checks",
          "entities",
          "extensions",
          "events",
          "filters",
```

```json
          "handlers",
          "hooks",
          "mutators",
          "silenced",
          "roles",
          "rolebindings"
        ],
        "resource_names": null
      },
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "namespaces"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "cluster-admin",
      "created_by": "admin"
    }
  }
]
```

API Specification

/clusterroles (GET)

description	Returns the list of cluster roles.
example url	http://hostname:8080/api/core/v2/clusterroles
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
| output | ```json
[
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "cluster-admin",
      "created_by": "admin"
    }
  }
]
``` |

Create a new cluster role

The `/clusterroles` API endpoint provides HTTP POST access to create a cluster role.

Example

In the following example, an HTTP POST request is submitted to the `/clusterroles` API endpoint to create a `global-event-reader` cluster role. The request includes the cluster role definition in the request body and returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterroles

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 | ```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)

Get a specific cluster role

The `/clusterroles/:clusterrole` API endpoint provides HTTP GET access to cluster role data for specific `:clusterrole` definitions, by cluster role `name`.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "global-event-reader",
    "created_by": "admin"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

API Specification

/clusterroles/:clusterrole (GET)	
description	Returns the specified cluster role.
example url	http://hostname:8080/api/core/v2/clusterroles/global-event-reader

response type	Map
response codes	⌐ **Success**: 200 (OK)
	⌐ **Missing**: 404 (Not Found)
	⌐ **Error**: 500 (Internal Server Error)
output	

```json
{
  "metadata": {
    "name": "global-event-reader",
    "created_by": "admin"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

Create or update a cluster role

The `/clusterroles/:clusterrole` API endpoint provides HTTP PUT access to create or update a cluster role, by cluster role name.

Example

In the following example, an HTTP PUT request is submitted to the `/clusterroles/:clusterrole`

API endpoint to update the `global-event-reader` cluster role by adding `"checks"` to the resources The request includes the cluster role definition in the request body and returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "checks",
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterroles


HTTP/1.1 201 Created
```

API Specification

/clusterroles/:clusterrole (PUT)	
description	Creates or updates the specified Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
payload	

```
{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

---

response codes

⌐ **Success**: 201 (Created)

⌐ **Malformed**: 400 (Bad Request)

⌐ **Error**: 500 (Internal Server Error)

Update a cluster role with PATCH

The `/clusterroles/:clusterrole` API endpoint provides HTTP PATCH access to update `:clusterrole` definitions, specified by cluster role name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/clusterroles/:clusterrole` API endpoint to update the verbs array within the rules array for the `global-event-admin` cluster role, resulting in an HTTP `200 OK` response and the updated cluster role definition.

We support [JSON merge patches](#), so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "rules": [
    {
      "verbs": [
        "*"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-admin

HTTP/1.1 200 OK
```

API Specification

/clusterroles/:clusterrole (PATCH)	
description	Updates the specified Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles/global-event-admin
payload	

```
{
  "rules": [
    {
      "verbs": [
        "*"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

response codes

- **Success**: 200 (OK)

- **Malformed**: 400 (Bad Request)

- **Error**: 500 (Internal Server Error)

Delete a cluster role

The `/clusterroles/:clusterrole` API endpoint provides HTTP DELETE access to delete a cluster role from Sensu (specified by the cluster role name).

Example

The following example shows a request to the `/clusterroles/:clusterrole` API endpoint to delete the cluster role `global-event-reader`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-reader

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

Get all datastore providers

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: Key $SENSU_API_KEY" \

HTTP/1.1 200 OK
[
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-other-postgres",
      "created_by": "admin"
    },
    "spec": {
      "batch_buffer": 0,
      "batch_size": 1,
      "batch_workers": 0,
      "dsn": "postgresql://user:secret@host:port/otherdbname",
      "max_conn_lifetime": "5m",
```

```
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true
    }
  },
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-postgres",
      "created_by": "admin"
    },
    "spec": {
      "dsn": "postgresql://user:secret@host:port/dbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true
    }
  }
]
```

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	```[{```

```json
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-postgres",
      "created_by": "admin"
    },
    "spec": {
      "batch_buffer": 0,
      "batch_size": 1,
      "batch_workers": 0,
      "dsn":
"postgresql://user:secret@host:port/otherdbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true
    }
  },
  {
    "type": "PostgresConfig",
    "api_version": "store/v1",
    "metadata": {
      "name": "my-postgres",
      "created_by": "admin"
    },
    "spec": {
      "dsn": "postgresql://user:secret@host:port/dbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true
    }
  }
]
```

Get a specific datastore provider

The `/provider/:provider` API endpoint provides HTTP PUT access to retrieve a Sensu datastore
provider.

Example

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
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",
    "created_by": "admin"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true
  }
}
```

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",
    "created_by": "admin"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn":
"postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true
  }
}
```

Create or update a datastore provider

The `/provider/:provider` API endpoint provides HTTP PUT access to create or update a Sensu datastore provider.

Example

```
curl -X PUT \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres \
-H "Authorization: Key $SENSU_API_KEY" \
-d '{
```

```
  "type": "PostgresConfig",
  "api_version": "store/v1",
  "metadata": {
    "name": "my-postgres"
  },
  "spec": {
    "batch_buffer": 0,
    "batch_size": 1,
    "batch_workers": 0,
    "dsn": "postgresql://user:secret@host:port/dbname",
    "max_conn_lifetime": "5m",
    "max_idle_conns": 2,
    "pool_size": 20,
    "strict": true
  }
}'

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": {
"batch_buffer": 0,
"batch_size": 1,
"batch_workers": 0,
``

```
      "dsn":
"postgresql://user:secret@host:port/dbname",
      "max_conn_lifetime": "5m",
      "max_idle_conns": 2,
      "pool_size": 20,
      "strict": true
    }
  }
```

response codes

- **Success**: 200 (OK)

- **Missing**: 404 (Not Found)

- **Error**: 500 (Internal Server Error)

Delete a datastore provider

The `/provider/:provider` API endpoint provides HTTP DELETE access to remove a Sensu datastore provider.

Example

The following example shows a request to the `/provider/:provider` API endpoint to remove the Sensu datastore provider with the ID `my-postgres`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/store/v1/provider/my-postgres

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	

- ¬ **Success**: 204 (No Content)
- ¬ **Missing**: 404 (Not Found)
- ¬ **Error**: 500 (Internal Server Error)

Entities API

Get all entities

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: Key $SENSU_API_KEY"

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": [
          {
            "name": "lo",
```

```json
      "addresses": [
        "127.0.0.1/8",
        "::1/128"
      ]
    },
    {
      "name": "enp0s3",
      "mac": "08:00:27:11:ad:d2",
      "addresses": [
        "10.0.2.15/24",
        "fe80::f50c:b029:30a5:3e26/64"
      ]
    },
    {
      "name": "enp0s8",
      "mac": "08:00:27:9f:5d:f3",
      "addresses": [
        "172.28.128.3/24",
        "fe80::a00:27ff:fe9f:5df3/64"
      ]
    }
  ]
},
"arch": "amd64",
"libc_type": "glibc",
"vm_system": "kvm",
"vm_role": "host",
"cloud_provider": "",
"processes": [
  {
    "name": "Slack",
    "pid": 1349,
    "ppid": 0,
    "status": "Ss",
    "background": true,
    "running": true,
    "created": 1582137786,
    "memory_percent": 1.09932518,
    "cpu_percent": 0.3263987595984941
  },
  {
    "name": "Slack Helper",
```

```json
          "pid": 1360,
          "ppid": 1349,
          "status": "Ss",
          "background": true,
          "running": true,
          "created": 1582137786,
          "memory_percent": 0.146866455,
          "cpu_percent": 0.308976181461092553
        }
      ]
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    }
  }
]
```

API Specification

/entities (GET)

description	Returns the list of entities.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)

output	<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": {
	        "interfaces": [
	          {
	            "name": "lo",
	            "addresses": [
	              "127.0.0.1/8",
	              "::1/128"
	            ]
	          },
	          {
	            "name": "enp0s3",
	            "mac": "08:00:27:11:ad:d2",
	            "addresses": [</pre>

```json
          "10.0.2.15/24",
          "fe80::f50c:b029:30a5:3e26/64"
        ]
      },
      {
        "name": "enp0s8",
        "mac": "08:00:27:9f:5d:f3",
        "addresses": [
          "172.28.128.3/24",
          "fe80::a00:27ff:fe9f:5df3/64"
        ]
      }
    ]
  },
  "arch": "amd64",
  "libc_type": "glibc",
  "vm_system": "kvm",
  "vm_role": "host",
  "cloud_provider": "",
  "processes": [
    {
      "name": "Slack",
      "pid": 1349,
      "ppid": 0,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 1.09932518,
      "cpu_percent": 0.3263987595984941
    },
    {
      "name": "Slack Helper",
      "pid": 1360,
      "ppid": 1349,
      "status": "Ss",
      "background": true,
      "running": true,
      "created": 1582137786,
      "memory_percent": 0.146866455,
      "cpu_percent": 0.308976181461092553
    }
```

```json
      ]
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    }
  }
]
```

Create a new entity

The `/entities` API endpoint provides HTTP POST access to create a Sensu entity.

Example

In the following example, an HTTP POST request is submitted to the `/entities` API endpoint to create a proxy entity named `sensu-centos`. The request includes the entity definition in the request

body and returns a successful `HTTP 200 OK` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity_class": "proxy",
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "web"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities

HTTP/1.1 200 OK
```

API Specification

/entities (POST)	
description	Creates a Sensu entity.
example URL	http://hostname:8080/api/core/v2/namespaces/default/entities

payload	
	``
	{
	  "entity_class": "proxy",
	  "sensu_agent_version": "1.0.0",
	  "subscriptions": [
	    "web"
	  ],
	``

```
            "deregister": false,
            "deregistration": {},
            "metadata": {
              "name": "sensu-centos",
              "namespace": "default",
              "labels": null,
              "annotations": null
            }
          }
```

response codes

¬ **Success**: 200 (OK)

¬ **Malformed**: 400 (Bad Request)

¬ **Error**: 500 (Internal Server Error)

Get a specific entity

The `/entities/:entity` API endpoint provides HTTP GET access to entity data for specific `:entity` definitions, by entity `name`.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "entity_class": "agent",
  "sensu_agent_version": "1.0.0",
  "system": {
```

```json
    "hostname": "sensu-centos",
    "os": "linux",
    "platform": "centos",
    "platform_family": "rhel",
    "platform_version": "7.4.1708",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::f50c:b029:30a5:3e26/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:9f:5d:f3",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe9f:5df3/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
```

```json
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 0.146866455,
        "cpu_percent": 0.308976181461092553
      }
    ]
  },
  "subscriptions": [
    "entity:sensu-centos"
  ],
  "last_seen": 1543349936,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "created_by": "admin",
```

```
      "labels": null,
      "annotations": null
    }
  }
}
```

API Specification

/entities/:entity (GET)	
description	Returns the specified entity.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output	<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": {
	      "interfaces": [
	        {
	          "name": "lo",
	          "addresses": [
	            "127.0.0.1/8",
	            "::1/128"
	          ]</pre>

```
        },
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::f50c:b029:30a5:3e26/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:9f:5d:f3",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe9f:5df3/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "kvm",
    "vm_role": "host",
    "cloud_provider": "",
    "processes": [
      {
        "name": "Slack",
        "pid": 1349,
        "ppid": 0,
        "status": "Ss",
        "background": true,
        "running": true,
        "created": 1582137786,
        "memory_percent": 1.09932518,
        "cpu_percent": 0.3263987595984941
      },
      {
        "name": "Slack Helper",
        "pid": 1360,
        "ppid": 1349,
        "status": "Ss",
        "background": true,
```

```json
      "running": true,
      "created": 1582137786,
      "memory_percent": 0.146866455,
      "cpu_percent": 0.308976181461092553
    }
   ]
  },
  "subscriptions": [
    "entity:sensu-centos"
  ],
  "last_seen": 1543349936,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  }
}
```

Create or update an entity

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 200 OK` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity_class": "proxy",
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "web",
    "system"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/sensu-centos

HTTP/1.1 200 OK
```

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 | ```json
{
  "entity_class": "proxy",
  "sensu_agent_version": "1.0.0",
  "subscriptions": [
    "web",
    "system"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
``` |

response codes

- ꓶ **Success**: 200 (OK)

- ꓶ **Malformed**: 400 (Bad Request)

- ꓶ **Error**: 500 (Internal Server Error)

Update an entity with PATCH

The `/entities/:entity` API endpoint provides HTTP PATCH access to update **entity configuration attributes** in `:entity` definitions, specified by entity name:

- ꓶ `labels`

- ꓶ `annotations`

- ꓶ `created_by`

- ⅂ `entity_class`
- ⅂ `user`
- ⅂ `subscriptions`
- ⅂ `deregister`
- ⅂ `deregistration`
- ⅂ `redact`
- ⅂ `keepalive_handler`

> **NOTE**: *You cannot change a resource's* `name` *or* `namespace` *with a PATCH request.Use a* [PUT request](#) *instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/entities/:entity` API endpoint to add a label for the `sensu-centos` entity, resulting in an HTTP `200 OK` response and the updated entity definition.

We support [JSON merge patches](#), so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/entities/sensu-centos

HTTP/1.1 200 OK
```

API Specification

/entities/:entity (PATCH)	
description	Updates the specified Sensu entity.
example URL	http://hostname:8080/api/core/v2/entities/sensu-centos
| payload | ```json
{
  "metadata": {
    "labels": {
      "region": "us-west-1"
    }
  }
}
``` |
| response codes | ⌐ **Success**: 200 (OK)
⌐ **Malformed**: 400 (Bad Request)
⌐ **Error**: 500 (Internal Server Error) |

Delete an entity

The `/entities/:entity` API endpoint provides HTTP DELETE access to delete an entity from Sensu (specified by the entity name).

Example

The following example shows a request to the `/entities/:entity` API endpoint to delete the entity `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: Key $SENSU_API_KEY"


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

Get all events

The `/events` API endpoint provides HTTP GET access to event data.

Example

The following example demonstrates a request to the `/events` API endpoint, resulting in a JSON array that contains event definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "check": {
      "command": "check-cpu-usage -w 75 -c 90",
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 60,
      "low_flap_threshold": 0,
      "publish": true,
```

```
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2 nice=0.0 system=0.0 idle=99.8
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      },
      "secrets": null,
```

```json
      "is_silenced": false,
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::bc00:e2c8:1059:3868/64"
              ]
            },
            {
              "name": "eth1",
              "mac": "08:00:27:73:87:93",
              "addresses": [
                "172.28.128.57/24",
                "fe80::a00:27ff:fe73:8793/64"
              ]
            }
          ]
        },
        "arch": "amd64",
        "libc_type": "glibc",
        "vm_system": "vbox",
        "vm_role": "guest",
```

```json
        "cloud_provider": "",
        "processes": null
      },
      "subscriptions": [
        "system",
        "entity:server1"
      ],
      "last_seen": 1620313661,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "server1",
        "namespace": "default"
      },
      "sensu_agent_version": "6.1.0"
    },
    "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
    "metadata": {
      "namespace": "default"
    },
    "timestamp": 1620313666
  }
]
```

API Specification

/events (GET)

description	Returns the list of events.
example url	http://hostname:8080/api/core/v2/namespaces/default/events
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "check": {
      "command": "check-cpu-usage -w 75 -c 90",
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 60,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
```

```json
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2
nice=0.0 system=0.0 idle=99.8 iowait=0.0 irq=0.0
softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
```

```
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313661,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
```

```
                    "api_token",
                    "access_key",
                    "secret_key",
                    "private_key",
                    "secret"
                ],
                "metadata": {
                    "name": "server1",
                    "namespace": "default"
                },
                "sensu_agent_version": "6.1.0"
            },
            "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
            "metadata": {
                "namespace": "default"
            },
            "timestamp": 1620313666
        }
    ]
```

Create a new event

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 `201 Created` response and the event definition.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
```

```
      "metadata": {
        "name": "server1",
        "namespace": "default"
      }
    },
    "check": {
      "output": "Server error",
      "state": "failing",
      "status": 2,
      "handlers": ["slack"],
      "interval": 60,
      "metadata": {
        "name": "server-health"
      }
    }
  }' \
  http://127.0.0.1:8080/api/core/v2/namespaces/default/events


  HTTP/1.1 201 Created
```

To create useful, actionable events, we recommend adding check attributes like `status` (`0` for OK, `1` for warning, `2` for critical), `output` , and `handlers` to the attributes included in this example. For more information about event attributes and their available values, read the event specification.

For events created with this endpoint, the following attributes have the default value `0` unless you specify a different value for testing:

- ⌐ `executed`

- ⌐ `issued`

- ⌐ `last_seen`

- ⌐ `status`

The `last_ok` attribute will default to `0` even if you manually specify OK status in the request body.

The `sensu_agent_version` attribute will return with a null value for events created with this endpoint because these events are not created by an agent-executed check.

API Specification

/events (POST)

description	Creates a new Sensu event. To update an existing event, use the `/events` PUT endpoint. If you create a new event that references an entity that does not already exist, sensu-backend will automatically create a proxy entity in the same namespace when the event is published. If you create an event that references an existing entity but includes different information for entity attributes, Sensu **will not** make any changes to the existing entity's definition based on the event you create via the API. *NOTE: An agent cannot belong to, execute checks in, or create events in more than one namespace.*
example URL	http://hostname:8080/api/core/v2/namespaces/default/events
| payload | ```json
{
  "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**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Get event data for a specific entity

The `/events/:entity` API endpoint provides HTTP GET access to event data specific to an `:entity`, by entity `name`.

Example

In the following example, querying the `/events/:entity` API endpoint returns a list of Sensu events for the `server1` entity and a successful HTTP `200 OK` response.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1 \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "check": {
      "command": "check-cpu-usage -w 75 -c 90",
      "handlers": [],
      "high_flap_threshold": 0,
      "interval": 60,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [
        "check-cpu-usage"
      ],
      "subscriptions": [
        "system"
      ],
```

```
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 0,
      "round_robin": false,
      "duration": 5.052973881,
      "executed": 1620313661,
      "history": [
        {
          "status": 0,
          "executed": 1620313601
        },
        {
          "status": 0,
          "executed": 1620313661
        }
      ],
      "issued": 1620313661,
      "output": "CheckCPU TOTAL OK: total=0.2 user=0.2 nice=0.0 system=0.0 idle=99.8
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313661,
      "occurrences": 2,
      "occurrences_watermark": 2,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "scheduler": "memory"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
```

```json
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
```

```json
        ],
        "last_seen": 1620313661,
        "deregister": false,
        "deregistration": {},
        "user": "agent",
        "redact": [
          "password",
          "passwd",
          "pass",
          "api_key",
          "api_token",
          "access_key",
          "secret_key",
          "private_key",
          "secret"
        ],
        "metadata": {
          "name": "server1",
          "namespace": "default"
        },
        "sensu_agent_version": "6.1.0"
      },
      "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
      "metadata": {
        "namespace": "default"
      },
      "timestamp": 1620313666
    },
    {
      "check": {
        "handlers": [
          "keepalive"
        ],
        "high_flap_threshold": 0,
        "interval": 20,
        "low_flap_threshold": 0,
        "publish": false,
        "runtime_assets": null,
        "subscriptions": [],
        "proxy_entity_name": "",
        "check_hooks": null,
        "stdin": false,
```

```json
    "subdue": null,
  "ttl": 0,
  "timeout": 120,
  "round_robin": false,
  "executed": 1620313714,
  "history": [
    {
      "status": 0,
      "executed": 1620313314
    },
    {
      "status": 0,
      "executed": 1620313334
    },
    {
      "status": 0,
      "executed": 1620313354
    },
    {
      "...": 0,
      "...": 1620313374
    }
  ],
  "issued": 1620313714,
  "output": "Keepalive last sent from server1 at 2021-05-06 15:08:34 +0000 UTC",
  "state": "passing",
  "status": 0,
  "total_state_change": 0,
  "last_ok": 1620313714,
  "occurrences": 358,
  "occurrences_watermark": 358,
  "output_metric_format": "",
  "output_metric_handlers": null,
  "env_vars": null,
  "metadata": {
    "name": "keepalive",
    "namespace": "default"
  },
  "secrets": null,
  "is_silenced": false,
  "scheduler": "etcd"
},
```

```json
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::bc00:e2c8:1059:3868/64"
              ]
            },
            {
              "name": "eth1",
              "mac": "08:00:27:73:87:93",
              "addresses": [
                "172.28.128.57/24",
                "fe80::a00:27ff:fe73:8793/64"
              ]
            }
          ]
        },
        "arch": "amd64",
        "libc_type": "glibc",
        "vm_system": "vbox",
        "vm_role": "guest",
        "cloud_provider": "",
        "processes": null
      },
```

```json
      "subscriptions": [
        "system",
        "entity:server1"
      ],
      "last_seen": 1620313714,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "server1",
        "namespace": "default"
      },
      "sensu_agent_version": "6.1.0"
    },
    "id": "8717b1dc-47d2-4b73-a259-ee2645cadbf5",
    "metadata": {
      "namespace": "default"
    },
    "timestamp": 1620313714
  }
]
```

API Specification

/events/:entity (GET)
description	Returns a list of events for the specified entity.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1

pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output	
	``json
	[
	  {
	    "check": {
	      "command": "check-cpu-usage -w 75 -c 90",
	      "handlers": [],
	      "high_flap_threshold": 0,
	      "interval": 60,
	      "low_flap_threshold": 0,
	      "publish": true,
	      "runtime_assets": [
	        "check-cpu-usage"
	      ],
	      "subscriptions": [
	        "system"
	      ],
	      "proxy_entity_name": "",
	      "check_hooks": null,
	      "stdin": false,
	      "subdue": null,
	      "ttl": 0,
	      "timeout": 0,
	      "round_robin": false,
	      "duration": 5.052973881,
	      "executed": 1620313661,
	      "history": [
	        {
	          "status": 0,
	          "executed": 1620313601
	        },
	``

```json
      {
        "status": 0,
        "executed": 1620313661
      }
    ],
    "issued": 1620313661,
    "output": "CheckCPU TOTAL OK: total=0.2 user=0.2
nice=0.0 system=0.0 idle=99.8 iowait=0.0 irq=0.0
softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1620313661,
    "occurrences": 2,
    "occurrences_watermark": 2,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check-cpu",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
    "scheduler": "memory"
  },
  "entity": {
    "entity_class": "agent",
    "system": {
      "hostname": "server1",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
```

```json
        },
        {
          "name": "eth0",
          "mac": "08:00:27:8b:c9:3f",
          "addresses": [
            "10.0.2.15/24",
            "fe80::bc00:e2c8:1059:3868/64"
          ]
        },
        {
          "name": "eth1",
          "mac": "08:00:27:73:87:93",
          "addresses": [
            "172.28.128.57/24",
            "fe80::a00:27ff:fe73:8793/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "system",
    "entity:server1"
  ],
  "last_seen": 1620313661,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
```

```json
        "private_key",
        "secret"
      ],
      "metadata": {
        "name": "server1",
        "namespace": "default"
      },
      "sensu_agent_version": "6.1.0"
    },
    "id": "da53be74-be42-4862-a481-b7e3236e8e6d",
    "metadata": {
      "namespace": "default"
    },
    "timestamp": 1620313666
  },
  {
    "check": {
      "handlers": [
        "keepalive"
      ],
      "high_flap_threshold": 0,
      "interval": 20,
      "low_flap_threshold": 0,
      "publish": false,
      "runtime_assets": null,
      "subscriptions": [],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "subdue": null,
      "ttl": 0,
      "timeout": 120,
      "round_robin": false,
      "executed": 1620313714,
      "history": [
        {
          "status": 0,
          "executed": 1620313314
        },
        {
          "status": 0,
          "executed": 1620313334
```

```
        },
        {
          "status": 0,
          "executed": 1620313354
        },
        {
          "...": 0,
          "...": 1620313374
        }
      ],
      "issued": 1620313714,
      "output": "Keepalive last sent from server1 at
2021-05-06 15:08:34 +0000 UTC",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313714,
      "occurrences": 358,
      "occurrences_watermark": 358,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "keepalive",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "scheduler": "etcd"
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
```

```json
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "eth0",
            "mac": "08:00:27:8b:c9:3f",
            "addresses": [
              "10.0.2.15/24",
              "fe80::bc00:e2c8:1059:3868/64"
            ]
          },
          {
            "name": "eth1",
            "mac": "08:00:27:73:87:93",
            "addresses": [
              "172.28.128.57/24",
              "fe80::a00:27ff:fe73:8793/64"
            ]
          }
        ]
      },
      "arch": "amd64",
      "libc_type": "glibc",
      "vm_system": "vbox",
      "vm_role": "guest",
      "cloud_provider": "",
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313714,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
```

```
            "api_key",
            "api_token",
            "access_key",
            "secret_key",
            "private_key",
            "secret"
          ],
          "metadata": {
            "name": "server1",
            "namespace": "default"
          },
          "sensu_agent_version": "6.1.0"
        },
        "id": "8717b1dc-47d2-4b73-a259-ee2645cadbf5",
        "metadata": {
          "namespace": "default"
        },
        "timestamp": 1620313714
      }
    ]
```

Get event data for a specific entity and check

The `/events/:entity/:check` API endpoint provides HTTP GET access to event data for the specified entity and check.

Example

In the following example, an HTTP GET request is submitted to the `/events/:entity/:check` API endpoint to retrieve the event for the `server1` entity and the `check-cpu` check.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/check-cpu \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
```

```json
{
  "check": {
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 5.050929017,
    "executed": 1620313539,
    "history": null,
    "issued": 1620313539,
    "output": "CheckCPU TOTAL OK: total=2.85 user=2.65 nice=0.0 system=0.2 idle=97.15 iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0 guest_nice=0.0\n",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1620313539,
    "occurrences": 1,
    "occurrences_watermark": 1,
    "output_metric_format": "",
    "output_metric_handlers": null,
    "env_vars": null,
    "metadata": {
      "name": "check-cpu",
      "namespace": "default"
    },
    "secrets": null,
    "is_silenced": false,
```

```json
      "scheduler": ""
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::bc00:e2c8:1059:3868/64"
              ]
            },
            {
              "name": "eth1",
              "mac": "08:00:27:73:87:93",
              "addresses": [
                "172.28.128.57/24",
                "fe80::a00:27ff:fe73:8793/64"
              ]
            }
          ]
        },
        "arch": "amd64",
        "libc_type": "glibc",
        "vm_system": "vbox",
        "vm_role": "guest",
        "cloud_provider": "",
```

```
      "processes": null
    },
    "subscriptions": [
      "system",
      "entity:server1"
    ],
    "last_seen": 1620313539,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "server1",
      "namespace": "default"
    },
    "sensu_agent_version": "6.1.0"
  },
  "id": "9a9c7515-0a04-43f3-9351-d8da88942b1b",
  "metadata": {
    "namespace": "default"
  },
  "timestamp": 1620313546
}
```

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/check-cpu
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	

```
{
  "check": {
    "command": "check-cpu-usage -w 75 -c 90",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 60,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": [
      "check-cpu-usage"
    ],
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "subdue": null,
    "ttl": 0,
    "timeout": 0,
    "round_robin": false,
    "duration": 5.050929017,
    "executed": 1620313539,
    "history": null,
    "issued": 1620313539,
    "output": "CheckCPU TOTAL OK: total=2.85
user=2.65 nice=0.0 system=0.2 idle=97.15
iowait=0.0 irq=0.0 softirq=0.0 steal=0.0 guest=0.0
```

```
guest_nice=0.0\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1620313539,
      "occurrences": 1,
      "occurrences_watermark": 1,
      "output_metric_format": "",
      "output_metric_handlers": null,
      "env_vars": null,
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      },
      "secrets": null,
      "is_silenced": false,
      "scheduler": ""
    },
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "server1",
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804",
        "network": {
          "interfaces": [
            {
              "name": "lo",
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ]
            },
            {
              "name": "eth0",
              "mac": "08:00:27:8b:c9:3f",
              "addresses": [
                "10.0.2.15/24",
                "fe80::bc00:e2c8:1059:3868/64"
              ]
```

```json
        },
        {
          "name": "eth1",
          "mac": "08:00:27:73:87:93",
          "addresses": [
            "172.28.128.57/24",
            "fe80::a00:27ff:fe73:8793/64"
          ]
        }
      ]
    },
    "arch": "amd64",
    "libc_type": "glibc",
    "vm_system": "vbox",
    "vm_role": "guest",
    "cloud_provider": "",
    "processes": null
  },
  "subscriptions": [
    "system",
    "entity:server1"
  ],
  "last_seen": 1620313539,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
    "name": "server1",
    "namespace": "default"
  },
  "sensu_agent_version": "6.1.0"
```

```
    },
    "id": "9a9c7515-0a04-43f3-9351-d8da88942b1b",
    "metadata": {
      "namespace": "default"
    },
    "timestamp": 1620313546
  }
```

Create a new event for an entity and check

The `/events/:entity/:check` API endpoint provides HTTP POST access to create an event and send it to the Sensu 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: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
```

```
      "name": "server-health"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health


HTTP/1.1 201 Created
```

> **NOTE**: A namespace is not required to create the event. The event will use the namespace in the URL by default.

You can use sensuctl or the Sensu web UI to view the event:

```
sensuctl event list
```

The response should list the event with the status and output specified in the request:

```
Entity     Check          Output           Status  Silenced       Timestamp
─────────────────────────────────────────────────────────────────────────────── ──────── ──────────
────────────────────────────────────────
 server1    server-health  Server error         1     false     2019-03-14 16:56:09 +0000 UTC
```

For events created with this endpoint, the following attributes have the default value `0` unless you specify a different value for testing:

- `executed`

- `issued`

- `last_seen`

- `status`

The `last_ok` attribute will default to `0` even if you manually specify OK status in the request body.

The `sensu_agent_version` attribute will return with a null value for events created with this endpoint because these events are not created by an agent-executed check.

API Specification

/events/:entity/:check (POST)

description	Creates an event for the specified entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1/server-health
payload	

```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**: 201 (Created)
- ¬ **Missing**: 404 (Not Found)
- ¬ **Error**: 500 (Internal Server Error)

Create or update an event for an entity and check

The `/events/:entity/:check` API endpoint provides HTTP PUT access to create or update an event and send it to the Sensu 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: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health


HTTP/1.1 201 Created
```

NOTE: *A namespace is not required to create the event. The event will use the namespace in the URL by default.*

You can use sensuctl or the <u>Sensu web UI</u> to view the event:

```
sensuctl event list
```

The response should list the event with the status and output specified in the request:

Entity	Check	Output	Status	Silenced	Timestamp
server1	server-health	Server error	1	false	2019-03-14 16:56:09 +0000 UTC

API Specification

/events/:entity/:check (PUT)	
description	Creates an event for the specified entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1/server-health
| payload | ```
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
``` |

```
      }
    }
```

payload parameters	Review the payload parameters section below.
response codes	¬ **Success**: 201 (Created) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Payload parameters

The `/events/:entity/:check` PUT endpoint requires a request payload that contains an `entity` scope and a `check` scope.

- ¬ The `entity` scope contains information about the component of your infrastructure represented by the event. At minimum, Sensu requires the `entity` scope to contain the `entity_class` (`agent` or `proxy`) and the entity `name` and `namespace` within a `metadata` scope. For more information about entity attributes, review the entity specification.

- ¬ The `check` scope contains information about the event status and how the event was created. At minimum, Sensu requires the `check` scope to contain a `name` within a `metadata` scope and either an `interval` or `cron` attribute. For more information about check attributes, review the check specification.

Example request with minimum required event attributes

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1"
    }
  },
  "check": {
```

```
      "interval": 60,
      "metadata": {
        "name": "server-health"
      }
    }
  }' \
  http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-health
```

The 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, review the event specification.

Example request with minimum recommended event attributes

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "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.Read the events reference and for more information about Sensu metric format.

Example request including metrics

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "status": 0,
    "output_metric_handlers": ["influxdb"],
    "interval": 60,
    "metadata": {
      "name": "server-metrics"
    }
  },
  "metrics": {
    "handlers": [
      "influxdb"
    ],
    "points": [
      {
        "name": "server1.server-metrics.time_total",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "server1.server-metrics.time_namelookup",
        "tags": [],
```

```
      "timestamp": 1552506033,
      "value": 0.004
    }
  ]
}
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-metrics
```

Delete an event

Example

The following example shows a request to the `/events/:entity/:check` API endpoint to delete the event produced by the `server1` entity and `check-cpu` check, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/check-cpu \
-H "Authorization: Key $SENSU_API_KEY"

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/server1/check-cpu
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found)

¬ **Error**: 500 (Internal Server Error)

Federation API

NOTE: Requests to the federation API require you to authenticate with a Sensu _API key_ or _access token_. The code examples in this document use the _environment variable_ `$SENSU_API_KEY` to represent a valid API key in API requests.

Get all replicators

The `/etcd-replicators` API endpoint provides HTTP GET access to a list of replicators.

NOTE: The etcd-replicators datatype is only accessible for users who have a cluster role that permits access to replication resources.

Example

The following example demonstrates a 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: Key $SENSU_API_KEY"
[
  {
    "api_version": "federation/v1",
    "type": "EtcdReplicator",
    "metadata": {
      "name": "my_replicator",
      "created_by": "admin"
```

```
    },
    "spec": {
      "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
      "cert": "/path/to/ssl/cert.pem",
      "key": "/path/to/ssl/key.pem",
      "insecure": false,
      "url": "http://remote-etcd.example.com:2379",
      "api_version": "core/v2",
      "resource": "Role",
      "replication_interval_seconds": 30
    }
  }
]
```

API Specification

/etcd-replicators (GET)	
description	Returns the list of replicators.
example url	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)
output	```[{ "api_version": "federation/v1", "type": "EtcdReplicator", "metadata": { "name": "my_replicator", "created_by": "admin" }, "spec": { "ca_cert": "/path/to/ssl/trusted-certificate-```

```
                    authorities.pem",
                "cert": "/path/to/ssl/cert.pem",
                "key": "/path/to/ssl/key.pem",
                "insecure": false,
                "url": "http://remote-etcd.example.com:2379",
                "api_version": "core/v2",
                "resource": "Role",
                "replication_interval_seconds": 30
            }
        }
    ]
```

Create a new replicator

The `/etcd-replicators` API endpoint provides HTTP POST access to create replicators.

> **NOTE**: Create a replicator for each resource type you want to replicate. Replicating `namespace` resources will **not** replicate the resources that belong to those namespaces.

Example

The following example demonstrates a request to the `/etcd-replicators` API endpoint to create the replicator `my_replicator`.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "my_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
```

```
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://remote-etcd.example.com:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}' \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators


HTTP/1.1 200 OK
```

API Specification

/etcd-replicators (POST)	
description	Creates a new replicator (if none exists).
example URL	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators

payload	<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",
	    "insecure": false,
	    "url": "http://remote-etcd.example.com:2379",
	    "api_version": "core/v2",
	    "resource": "Role",
	    "replication_interval_seconds": 30
	  }
	}</pre>

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific replicator

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP GET access to data for a specific `:etcd-replicator`, by replicator name.

> *NOTE: The etcd-replicators datatype is only accessible for users who have a cluster role that permits access to replication resources.*

Example

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: Key $SENSU_API_KEY"
{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "my_replicator",
    "created_by": "admin"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
```

```
      "url": "http://remote-etcd.example.com:2379",
      "api_version": "core/v2",
      "resource": "Role",
      "replication_interval_seconds": 30
    }
  }
```

API Specification

/etcd-replicators/:etcd-replicator (GET)	
description	Returns the specified replicator.
example url	http://hostname:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	```{
"api_version": "federation/v1",
"type": "EtcdReplicator",
"metadata": {
"name": "my_replicator",
"created_by": "admin"
},
"spec": {
"ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
"cert": "/path/to/ssl/cert.pem",
"key": "/path/to/ssl/key.pem",
"insecure": false,
"url": "http://remote-etcd.example.com:2379",```

```
      "api_version": "core/v2",
      "resource": "Role",
      "replication_interval_seconds": 30
    }
  }
```

Create or update a replicator

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP PUT access to create or update a specific `:etcd-replicator`, by replicator name.

Example

The following example demonstrates a request to the `/etcd-replicators/:etcd-replicator` API endpoint to update the replicator `my_replicator`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "api_version": "federation/v1",
  "type": "EtcdReplicator",
  "metadata": {
    "name": "my_replicator"
  },
  "spec": {
    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
    "cert": "/path/to/ssl/cert.pem",
    "key": "/path/to/ssl/key.pem",
    "insecure": false,
    "url": "http://remote-etcd.example.com:2379",
    "api_version": "core/v2",
    "resource": "Role",
    "replication_interval_seconds": 30
  }
}' \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my-replicator
```

```
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	
	``json
	{
	  "api_version": "federation/v1",
	  "type": "EtcdReplicator",
	  "metadata": {
	    "name": "my_replicator"
	  },
	  "spec": {
	    "ca_cert": "/path/to/ssl/trusted-certificate-authorities.pem",
	    "cert": "/path/to/ssl/cert.pem",
	    "key": "/path/to/ssl/key.pem",
	    "insecure": false,
	    "url": "http://remote-etcd.example.com:2379",
	    "api_version": "core/v2",
	    "resource": "Role",
	    "replication_interval_seconds": 30
	  }
	}
	``

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request)

¬ **Error**: 500 (Internal Server Error)

Delete a replicator

The `/etcd-replicators/:etcd-replicator` API endpoint provides HTTP DELETE access to delete the specified replicator from Sensu.

Example

The following example shows a request to the `/etcd-replicators/:etcd-replicator` API endpoint to delete the replicator `my_replicator`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/federation/v1/etcd-replicators/my_replicator


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)

Get all clusters

The `/clusters` API endpoint provides HTTP GET access to a list of clusters.

Example

The following example demonstrates a request to the `/clusters` API endpoint, resulting in a list of clusters.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK

[
    {
        "type": "Cluster",
        "api_version": "federation/v1",
        "metadata": {
            "name": "us-west-2a",
            "created_by": "admin"
        },
        "spec": {
            "api_urls": [
                "http://10.0.0.1:8080",
                "http://10.0.0.2:8080",
                "http://10.0.0.3:8080"
            ]
        }
    }
]
```

API Specification

/clusters (GET)

description	Returns the list of clusters.
example url	http://hostname:8080/api/enterprise/federation/v1/clusters
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)
output	

```json
[
    {
        "type": "Cluster",
        "api_version": "federation/v1",
        "metadata": {
            "name": "us-west-2a",
            "created_by": "admin"
        },
        "spec": {
            "api_urls": [
                "http://10.0.0.1:8080",
                "http://10.0.0.2:8080",
                "http://10.0.0.3:8080"
            ]
        }
    }
]
```

Get a specific cluster

The `/clusters/:cluster` API endpoint provides HTTP GET access to data for a specific `cluster`, by cluster name.

Example

In the following example, querying the `/clusters/:cluster` API endpoint returns a JSON map that

contains the requested `:cluster` .

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK

{
  "type": "Cluster",
  "api_version": "federation/v1",
  "metadata": {
      "name": "us-west-2a",
      "created_by": "admin"
  },
  "spec": {
      "api_urls": [
          "http://10.0.0.1:8080",
          "http://10.0.0.2:8080",
          "http://10.0.0.3:8080"
      ]
  }
}
```

API Specification

/clusters/:cluster (GET)

description	Returns the specified cluster.
example url	http://hostname:8080/api/enterprise/federation/v1/clusters/us-west-2a
response type	Map
response codes	┐ **Success**: 200 (OK) ┐ **Missing**: 404 (Not Found)

⌐ **Error**: 500 (Internal Server Error)

output

```json
{
    "type": "Cluster",
    "api_version": "federation/v1",
    "metadata": {
        "name": "us-west-2a",
        "created_by": "admin"
    },
    "spec": {
        "api_urls": [
            "http://10.0.0.1:8080",
            "http://10.0.0.2:8080",
            "http://10.0.0.3:8080"
        ]
    }
}
```

Create or update a cluster

The `/clusters/:cluster` API endpoint provides HTTP PUT access to create or update a specific `cluster`, by cluster name.

NOTE: Only cluster admins have PUT access to clusters.

Example

The following example demonstrates a request to the `/clusters/:cluster` API endpoint to update the cluster `us-west-2a` .

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
```

```
    "type": "Cluster",
    "api_version": "federation/v1",
    "metadata": {
        "name": "us-west-2a"
    },
    "spec": {
        "api_urls": [
            "http://10.0.0.1:8080",
            "http://10.0.0.2:8080",
            "http://10.0.0.3:8080"
        ]
    }
}' \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a


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	<pre>{
	    "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"
	        ]
	    }</pre>

```
    }
```

---

response codes

- ¬ **Success**: 201 (Created)
- ¬ **Malformed**: 400 (Bad Request)
- ¬ **Error**: 500 (Internal Server Error)

Delete a cluster

The `/clusters/:cluster` API endpoint provides HTTP DELETE access to delete the specified cluster from Sensu.

> **NOTE**: Only cluster admins have DELETE access to clusters.

Example

The following example shows a request to the `/clusters/:cluster` API endpoint to delete the cluster `us-west-2a`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/federation/v1/clusters/us-west-2a

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

Get all event filters

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",
      "created_by": "admin"
    },
    "action": "deny",
    "expressions": [
      "event.entity.metadata.namespace == 'development'"
    ],
    "runtime_assets": null
  },
  {
```

```
    "metadata": {
      "name": "state_change_only",
      "namespace": "default"
    },
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": null
  }
]
```

API Specification

/filters (GET)	
description	Returns the list of event filters.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output	```[
	  {
	    "metadata": {
	      "name": "development_filter",
	      "namespace": "default",
	      "created_by": "admin"
	    },```

```
      "action": "deny",
      "expressions": [
        "event.entity.metadata.namespace == 'development'"
      ],
      "runtime_assets": null
    },
    {
      "metadata": {
        "name": "state_change_only",
        "namespace": "default"
      },
      "action": "allow",
      "expressions": [
        "event.check.occurrences == 1"
      ],
      "runtime_assets": null
    }
  ]
```

Create a new event filter

The `/filters` API endpoint provides HTTP POST access to create an event filter.

Example

In the following example, an HTTP POST request is submitted to the `/filters` API endpoint to create the event filter `development_filter`. The request returns a successful HTTP `201 Created` response

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "development_filter",
    "namespace": "default",
    "labels": null,
    "annotations": null
```

```
  },
  "action": "deny",
  "expressions": [
    "event.entity.metadata.namespace == 'development'"
  ],
  "runtime_assets": []
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters


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)

Get a specific event filter

The `/filters/:filter` API endpoint provides HTTP GET access to event filter data for specific `:filter` definitions, by filter name.

Example

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",
    "created_by": "admin"
  },
  "action": "allow",
  "expressions": [
    "event.check.occurrences == 1"
  ],
  "runtime_assets": null
}
```

API Specification

/filters/:filter (GET)	
description	Returns the specified event filter.

example url	http://hostname:8080/api/core/v2/namespaces/default/filters/state_change_only
response type	Map
response codes	⌐ **Success**: 200 (OK) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)
| output | ```json
{
  "metadata": {
    "name": "state_change_only",
    "namespace": "default",
    "created_by": "admin"
  },
  "action": "allow",
  "expressions": [
    "event.check.occurrences == 1"
  ],
  "runtime_assets": null
}
``` |

Create or update an event filter

The `/filters/:filter` API endpoint provides HTTP PUT access to create or update an event filter.

Example

In the following example, an HTTP PUT request is submitted to the `/filters` API endpoint to create the event filter `development_filter`. The request returns a successful HTTP `200 OK` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
```

```
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "development_filter",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "action": "deny",
  "expressions": [
    "event.entity.metadata.namespace == 'development'"
  ],
  "runtime_assets": []
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/development_filter

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)

Update a filter with PATCH

The `/filters/:filter` API endpoint provides HTTP PATCH access to update `:filter` definitions, specified by filter name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/filters/:filter` API endpoint to update the expressions array for the `us-west` filter, resulting in an HTTP `200 OK` response and the updated filter definition.

We support _JSON merge patches_, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "expressions": [
    "event.check.occurrences == 3"
  ]
```

```
  }' \
  http://127.0.0.1:8080/api/core/v2/namespaces/default/filter/us-west


  HTTP/1.1 200 OK
```

API Specification

/filters/:filter (PATCH)	
description	Updates the specified Sensu filter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/filter/us-west

payload	<pre>{
	  "expressions": [
	    "event.check.occurrences == 3"
	  ]
	}</pre>

response codes	¬ **Success**: 200 (OK) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Delete an event filter

The `/filters/:filter` API endpoint provides HTTP DELETE access to delete an event filter from Sensu (specified by the filter name).

Example

The following example shows a request to the `/filters/:filter` API endpoint to delete the event filter `development_filter`, 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: Key $SENSU_API_KEY"


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

Get all handlers

The `/handlers` API endpoint provides HTTP GET access to <u>handler</u> data.

Example

The following example demonstrates a request to the `/handlers` API endpoint, resulting in a JSON array that contains <u>handler definitions</u>.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "influx-db",
      "namespace": "default",
      "created_by": "admin"
    },
    "type": "pipe",
    "command": "sensu-influxdb-handler -d sensu",
    "timeout": 0,
    "handlers": null,
    "filters": null,
    "env_vars": [
      "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
```

```
    "INFLUXDB_USER=sensu",
    "INFLUXDB_PASSWORD=password"
  ],
  "runtime_assets": ["sensu/sensu-influxdb-handler"]
},
{
  "metadata": {
    "name": "slack",
    "namespace": "default",
    "created_by": "admin"
  },
  "type": "pipe",
  "command": "sensu-slack-handler --channel '#monitoring'",
  "timeout": 0,
  "handlers": null,
  "filters": [
    "is_incident",
    "not_silenced"
  ],
  "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXXX"
  ],
  "runtime_assets": ["sensu/sensu-influxdb-handler"]
}
]
```

API Specification

/handlers (GET)	
description	Returns the list of handlers.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.

response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)

output	`[`
	`  {`
	`    "metadata": {`
	`      "name": "influx-db",`
	`      "namespace": "default",`
	`      "created_by": "admin"`
	`    },`
	`    "type": "pipe",`
	`    "command": "sensu-influxdb-handler -d sensu",`
	`    "timeout": 0,`
	`    "handlers": null,`
	`    "filters": null,`
	`    "env_vars": [`
	
	`"INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",`
	`      "INFLUXDB_USER=sensu",`
	`      "INFLUXDB_PASSWORD=password"`
	`    ],`
	`    "runtime_assets": ["sensu/sensu-influxdb-handler"]`
	`  },`
	`  {`
	`    "metadata": {`
	`      "name": "slack",`
	`      "namespace": "default"`
	`    },`
	`    "type": "pipe",`
	`    "command": "sensu-slack-handler --channel`
	`'#monitoring'",`
	`    "timeout": 0,`
	`    "handlers": null,`
	`    "filters": [`
	`      "is_incident",`
	`      "not_silenced"`

```
      ],
      "env_vars": [

      "SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T000000
      00/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
      ],
      "runtime_assets": ["sensu/sensu-slack-handler"]
    }
  ]
```

Create a new handler

The `/handlers` API endpoint provides HTTP POST access to create a handler.

Example

In the following example, an HTTP POST request is submitted to the `/handlers` API endpoint to create the event handler `influx-db`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "influx-db",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-influxdb-handler -d sensu",
  "env_vars": [
    "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
    "INFLUXDB_USER=sensu",
    "INFLUXDB_PASSWORD=password"
  ],
  "filters": [],
```

```
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers

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	<pre>{
	  "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"
	}</pre>

response codes

- ¬ **Success**: 201 (Created)
- ¬ **Malformed**: 400 (Bad Request)
- ¬ **Error**: 500 (Internal Server Error)

Get a specific handler

The `/handlers/:handler` API endpoint provides HTTP GET access to handler data for specific `:handler` definitions, by handler name.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "slack",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-slack-handler --channel '#monitoring'",
  "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
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	<pre>{
	  "metadata": {
	    "name": "slack",
	    "namespace": "default",
	    "created_by": "admin",
	    "labels": null,
	    "annotations": null
	  },
	  "command": "sensu-slack-handler --channel
	'#monitoring'",
	  "env_vars": [
	
	"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T</pre>

```
00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
  ],
  "filters": [
    "is_incident",
    "not_silenced"
  ],
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
}
```

Create or update a handler

The `/handlers/:handler` API endpoint provides HTTP PUT access to create or update a specific `:handler` definition, by handler name.

Example

In the following example, an HTTP PUT request is submitted to the `/handlers/:handler` API endpoint to create the handler `influx-db`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "influx-db",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-influxdb-handler -d sensu",
  "env_vars": [
    "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
    "INFLUXDB_USER=sensu",
```

```
      "INFLUXDB_PASSWORD=password"
  ],
  "filters": [],
  "handlers": [],
  "runtime_assets": ["sensu/sensu-influxdb-handler"],
  "timeout": 0,
  "type": "pipe"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/influx-db


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	<pre>{
	  "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": [],</pre>

```
      "timeout": 0,
      "type": "pipe"
  }
```

---

response codes

- ¬ **Success**: 201 (Created)
- ¬ **Malformed**: 400 (Bad Request)
- ¬ **Error**: 500 (Internal Server Error)

Update a handler with PATCH

The `/handlers/:handler` API endpoint provides HTTP PATCH access to update `:handler` definitions, specified by handler name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request.Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/handlers/:handler` API endpoint to update the filters array for the `influx-db` handler, resulting in an HTTP `200 OK` response and the updated handler definition.

We support _JSON merge patches_, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "filters": [
```

```
    "us-west",
    "is_incident"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/influx-db


HTTP/1.1 200 OK
```

API Specification

/handlers/:handler (PATCH)	
description	Updates the specified Sensu handler.
example URL	http://hostname:8080/api/core/v2/namespaces/default/handlers/influx-db

payload	<code>{
	  "filters": [
	    "us-west",
	    "is_incident"
	  ]
	}</code>

response codes

- ¬ **Success**: 200 (OK)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Delete a handler

The `/handlers/:handler` API endpoint provides HTTP DELETE access to delete a handler from Sensu (specified by the handler name).

Example

The following example shows a request to the `/handlers/:handler` API endpoint to delete the handler `slack`, 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: Key $SENSU_API_KEY"


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

Get health data

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
  },
  "PostgresHealth": [
    {
      "Name": "my-postgres",
      "Active": false,
```

```
      "Healthy": false
    }
  ]
}
```

API Specification

<table>
<tr>
<td colspan="2">/health (GET)</td>
</tr>
<tr>
<td>description</td>
<td>Returns health information about the Sensu instance.</td>
</tr>
<tr>
<td>example url</td>
<td>http://hostname:8080/health</td>
</tr>
<tr>
<td>query parameters</td>
<td>`timeout` : Defines the timeout when querying etcd. Default is `3` .</td>
</tr>
<tr>
<td>response type</td>
<td>Map</td>
</tr>
<tr>
<td>response codes</td>
<td>

⌐ **Success**: 200 (OK)

⌐ **Error**: 400 (Bad Request)

NOTE: The HTTP response codes for the health endpoint indicate whether your request reached Sensu rather than the health of your Sensu instance. To determine the health of your Sensu instance, you must process the JSON response body for your request. The <u>health specification</u> describes each attribute in the response body.

</td>
</tr>
<tr>
<td>output</td>
<td>

```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 2882886652148554927,
      "MemberIDHex": "8923110df66458af",
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
```

</td>
</tr>
</table>

```
    ],
    "Header": {
      "cluster_id": 4255616344056076734,
      "member_id": 2882886652148554927,
      "raft_term": 26
    },
    "PostgresHealth": [
      {
        "Name": "my-postgres",
        "Active": false,
        "Healthy": false
      }
    ]
}
```

Hooks API

Get all hooks

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "nginx-log",
      "namespace": "default",
      "created_by": "admin"
    },
    "command": "tail -n 100 /var/log/nginx/error.log",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  },
  {
    "metadata": {
```

```
        "name": "process-tree",
        "namespace": "default",
        "created_by": "admin"
    },
    "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu | head -n 6",
    "timeout": 10,
    "stdin": false,
    "runtime_assets": null
  }
]
```

API Specification

/hooks (GET)	
description	Returns the list of hooks.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "nginx-log",
      "namespace": "default",
      "created_by": "admin"
    },
    "command": "tail -n 100 /var/log/nginx/error.log",
```

```
        "timeout": 10,
        "stdin": false,
        "runtime_assets": null
      },
      {
        "metadata": {
          "name": "process-tree",
          "namespace": "default",
          "created_by": "admin"
        },
        "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu |
    head -n 6",
        "timeout": 10,
        "stdin": false,
        "runtime_assets": null
      }
    ]
```

Create a new hook

The `/hooks` API endpoint provides HTTP POST access to create a hook.

Example

In the following example, an HTTP POST request is submitted to the `/hooks` API endpoint to create the hook `process-tree` .The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
```

```
    "command": "ps -eo user,pid,cmd:50,%cpu --sort=-%cpu | head -n 6",
    "timeout": 10,
    "stdin": false
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks


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	
	``json
	{
	  "metadata": {
	    "name": "process-tree",
	    "namespace": "default",
	    "labels": null,
	    "annotations": null
	  },
	  "command": "ps aux",
	  "timeout": 10,
	  "stdin": false
	}
	``

response codes

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Get a specific hook

The `/hooks/:hook` API endpoint provides HTTP GET access to hook data for specific `:hook` definitions, by hook name.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "command": "ps aux",
  "timeout": 10,
  "stdin": false
}
```

API Specification

/hooks/:hook (GET)	
description	Returns the specified hook.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree
response type	Map

response codes

output

```json
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "command": "ps aux",
  "timeout": 10,
  "stdin": false
}
```

Create or update a hook

The `/hooks/:hook` API endpoint provides HTTP PUT access to create or update specific `:hook` definitions, by hook name.

Example

In the following example, an HTTP PUT request is submitted to the `/hooks/:hook` API endpoint to create the hook `nginx-log`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "nginx-log",
```

```
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "tail -n 100 /var/log/nginx/error.log",
  "timeout": 10,
  "stdin": false
  }' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/nginx-log


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)

Update a hook with PATCH

The `/hooks/:hook` API endpoint provides HTTP PATCH access to update `:hook` definitions, specified by hook name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/hooks/:hook` API endpoint to update the timeout for the `process-tree` hook, resulting in an HTTP `200 OK` response and the updated hook definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "timeout": 20
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hook/process-tree


HTTP/1.1 200 OK
```

API Specification

/hooks/:hook (PATCH)

description	Updates the specified Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree

payload	```json
	{
	  "timeout": 20
	}
	``

response codes	˥ **Success**: 200 (OK) ˥ **Malformed**: 400 (Bad Request) ˥ **Error**: 500 (Internal Server Error)

Delete a hook

The `/hooks/:hook` API endpoint provides HTTP DELETE access to delete a check hook from Sensu (specified by the hook name).

Example

The following example shows a request to the `/hooks/:hook` API endpoint to delete the hook `process-tree`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/process-tree \
-H "Authorization: Key $SENSU_API_KEY"


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 API

For more information about commercial features designed for enterprises, read [Get started with commercial features](#).

Get the active license configuration

The `/license` API endpoint provides HTTP GET access to the active license configuration.

Example

The following example demonstrates a 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/enterprise/licensing/v2/license \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json'

HTTP/1.1 200 OK
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
```

```
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
        "algorithm": "PSS",
        "hashAlgorithm": "SHA256",
        "saltLength": 20
      }
    },
    "signature": "XXXXXXXXXX",
    "metadata": {}
  }
}
```

API Specification

/license (GET)	
description	Returns the active commercial license configuration. To download your license, log in to your Sensu account or contact the Sensu sales team for a free trial.
example url	http://hostname:8080/api/enterprise/licensing/v2/license
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	

```json
{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
        "algorithm": "PSS",
        "hashAlgorithm": "SHA256",
        "saltLength": 20
      }
    },
    "signature": "XXXXXXXXXX",
    "metadata": {}
  }
}
```

Activate a commercial license

The `/license` API endpoint provides HTTP PUT access to activate a commercial license.

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

Example

In the following example, an HTTP PUT request is submitted to the `/license` API endpoint to create the license definition. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "LicenseFile",
  "api_version": "licensing/v2",
  "metadata": {
    "labels": {
      "sensu.io/entity-count": "10",
      "sensu.io/entity-limit": "100"
    }
  },
  "spec": {
    "license": {
      "version": 1,
      "issuer": "Sensu, Inc.",
      "accountName": "my_account",
      "accountID": 1234567,
      "issued": "2019-01-01T13:40:25-08:00",
      "validUntil": "2020-01-01T13:40:25-08:00",
      "plan": "managed",
      "features": [
        "all"
      ],
      "signature": {
        "algorithm": "PSS",
        "hashAlgorithm": "SHA256",
        "saltLength": 20
      }
    },
    "signature": "XXXXXXXXX",
    "metadata": {}
  }
}' \
```

```
http://127.0.0.1:8080/api/enterprise/licensing/v2/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": {
	    "labels": {
	      "sensu.io/entity-count": "10",
	      "sensu.io/entity-limit": "100"
	    }
	  },
	  "spec": {
	    "license": {
	      "version": 1,
	      "issuer": "Sensu, Inc.",
	      "accountName": "my_account",
	      "accountID": 1234567,
	      "issued": "2019-01-01T13:40:25-08:00",
	      "validUntil": "2020-01-01T13:40:25-08:00",
	      "plan": "managed",
	      "features": [
	        "all"
	      ],
	      "signature": {
	        "algorithm": "PSS",
	        "hashAlgorithm": "SHA256",
	        "saltLength": 20
	``

```
      }
    },
    "signature": "XXXXXXXXX",
    "metadata": {}
  }
}
```

response codes

- ¬ **Success**: 201 (Created)
- ¬ **Malformed**: 400 (Bad Request)
- ¬ **Error**: 500 (Internal Server Error)

Delete a commercial license

The `/license` API endpoint provides HTTP DELETE access to remove a commercial license.

Example

The following example shows a request to the `/license` API endpoint to delete the commercial license, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/enterprise/licensing/v2/license \
-H "Authorization: Key $SENSU_API_KEY"

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

Get Sensu metrics

The `/metrics` API endpoint provides HTTP GET access to internal Sensu metrics in [Prometheus](#) format, including embedded etcd, memory usage, garbage collection, and gRPC metrics.

Example

The following example demonstrates a 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

Get all mutators

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
  }
```

```
  ]
```

API Specification

<table>
<tr><td colspan="2">/mutators (GET)</td></tr>
<tr><td>description</td><td>Returns the list of mutators.</td></tr>
<tr><td>example url</td><td>http://hostname:8080/api/core/v2/namespaces/default/mutators</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>

```
[
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
  }
]
```

</td></tr>
</table>

Create a new mutator

The `/mutators` API endpoint provides HTTP POST access to create mutators.

Example

In the following example, an HTTP POST request is submitted to the `/mutators` API endpoint to create the mutator `example-mutator`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}' \
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

```json
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
```

response codes

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Get a specific mutator

The `/mutators/:mutator` API endpoint provides HTTP GET access to mutator data for specific `:mutator` definitions, by mutator name.

Example

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: Key $SENSU_API_KEY"
```

```
HTTP/1.1 200 OK
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "created_by": "admin",
    "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/example-mutator
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output	```{
	  "metadata": {
	    "name": "example-mutator",
	    "namespace": "default",
	    "created_by": "admin",
	    "labels": null,
	    "annotations": null```

```
        },
        "command": "example_mutator.go",
        "timeout": 0,
        "env_vars": [],
        "runtime_assets": []
    }
```

Create or update a mutator

The `/mutators/:mutator` API endpoint provides HTTP PUT access to mutator data to create or update specific `:mutator` definitions, by mutator name.

Example

In the following example, an HTTP PUT request is submitted to the `/mutators/:mutator` API endpoint to create the mutator `example-mutator`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}' \
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 | ```json
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
``` |
| response codes | ⌐ **Success**: 201 (Created)
⌐ **Malformed**: 400 (Bad Request)
⌐ **Error**: 500 (Internal Server Error) |

Update a mutator with PATCH

The `/mutators/:mutator` API endpoint provides HTTP PATCH access to update `:mutator` definitions, specified by mutator name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a `PUT`

request instead.

Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/mutators/:mutator` API endpoint to update the timeout for the `example-mutator` mutator, resulting in an HTTP `200 OK` response and the updated mutator definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "timeout": 10
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator


HTTP/1.1 200 OK
```

API Specification

/mutators/:mutator (PATCH)	
description	Updates the specified Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators/process-tree
payload	```{ "timeout": 10 }```

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Delete a mutator

The `/mutators/:mutator` API endpoint provides HTTP DELETE access to delete a mutator from Sensu (specified by the mutator name).

Example

The following example shows a request to the `/mutators/:mutator` API endpoint to delete the mutator `example-mutator`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator \
-H "Authorization: Key $SENSU_API_KEY" \

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

Get all namespaces

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: Key $SENSU_API_KEY"

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"
}
]```

Create a new namespace

The `/namespaces` API endpoint provides HTTP POST access to create Sensu namespaces.

Example

In the following example, an HTTP POST request is submitted to the `/namespaces` API endpoint to create the namespace `development`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces


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)

Create or update a namespace

The `/namespaces/:namespace` API endpoint provides HTTP PUT access to create or update specific Sensu namespaces, by namespace name.

Example

In the following example, an HTTP PUT request is submitted to the `/namespaces/:namespace` API endpoint to create the namespace `development`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "name": "development"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/development


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)

Delete a namespace

The `/namespaces/:namespace` API endpoint provides HTTP DELETE access to delete a namespace from Sensu (specified by the namespace name).

Example

The following example shows a request to the `/namespaces/:namespace` API endpoint to delete the namespace `development`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/development \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 204 No Content
```

Namespaces must be empty before you can delete them. If the response to your delete request includes `Error: resource is invalid: namespace is not empty`, the namespace may still contain events or other resources. To remove all resources and events so that you can delete a namespace, use this sensuctl dump command (replace `<namespace-name>` with the namespace you want to empty):

```
sensuctl dump entities,events,assets,checks,filters,handlers,secrets/v1.Secret --
namespace <namespace-name> | sensuctl delete
```

API Specification

/namespaces/:namespace (DELETE)	
description	Removes the specified namespace from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/development
response codes	⌐ **Success**: 204 (No Content)

- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

Get all namespaces for a specific user

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: Key $SENSU_API_KEY"

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	
| | ```json
[
  {
    "name": "default"
  },
  {
    "name": "development"
  }
]
``` |

Prune API

NOTE: The prune API is an alpha feature and may include breaking changes. The prune API requires *cluster-level privileges*, even when all resources belong to the same namespace.

Requests to the prune API require you to authenticate with a Sensu *API key* or *access token*. The code examples in this document use the *environment variable* `$SENSU_API_KEY` to represent a valid API key in API requests.

Create a new pruning command

The `/prune/v1alpha` API endpoint provides HTTP POST access to create a pruning command to delete resources that are not specified in the request body.

Example

In the following example, an HTTP POST request is submitted to the `/prune/v1alpha` API endpoint to create a pruning command for the checks specified in the request body in the `dev` namespace created by any user.

The request returns a successful HTTP `201 Created` response and a list of the resources that were pruned.

```
curl -X POST \
http://127.0.0.1:8080/api/enterprise/prune/v1alpha\?types\=core/v2.CheckConfig\&allUs
ers\=true\&namespaces\=dev \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "CheckConfig",
```

```
  "api_version": "core/v2",
  "name": "check-echo",
  "namespace": "dev",
  "labels": {
    "region": "us-west-2",
    "sensu.io/managed_by": "sensuctl"
  },
  "created_by": "admin"
}'


HTTP/1.1 201 Created

[
  {
    "type": "CheckConfig",
    "api_version": "core/v2",
    "name": "check-echo",
    "namespace": "dev",
    "labels": {
      "region": "us-west-2",
      "sensu.io/managed_by": "sensuctl"
    },
    "created_by": "admin"
  }
]
```

API Specification

/prune/v1alpha (POST)
description	Creates a pruning command to delete the specified resources.
example URL	http://hostname:8080/api/enterprise/prune/v1alpha

example payload	`{`
	`  "type": "CheckConfig",`
	`  "api_version": "core/v2",`
	`  "name": "check-echo",`

```
  "namespace": "dev",
  "labels": {
    "region": "us-west-2",
    "sensu.io/managed_by": "sensuctl"
  },
  "created_by": "admin"
}
```

query parameters

- `type` : The fully-qualified name of the resource you want to prune. Example: `?type=core/v2.CheckConfig` .

- `allUsers` : Prune resources created by all users. Mutually exclusive with the `users` parameter. Defaults to false. Example: `?allUsers=true` .

- `clusterWide` : Prune any cluster-wide (non-namespaced) resources that are not defined in the configuration. Defaults to false. Example: `?clusterWide=true` .

- `dryRun` : Print the resources that will be pruned but does not actually delete them. Defaults to false. Example: `?dryRun=true` .

- `labelSelector` : Prune only resources that match the specified labels (accepts multiple values). Labels are a commercial feature. Example: `?labelSelector=[...]` .

- `namespaces` : The namespace where you want to apply pruning. Example: `?namespaces=dev` .

- `users` : Prune only resources that were created by the specified users (accepts multiple values). Defaults to the currently configured sensuctl user. Example: `?users=admin` .

To use multiple values for the parameters that allow them, you must specify the parameter multiple times (for example, `?users=admin&users=dev`) rather than using a comma-separated list.

payload

```
[
  {
    "type": "CheckConfig",
```

```
    "api_version": "core/v2",
    "name": "check-echo",
    "namespace": "dev",
    "labels": {
      "region": "us-west-2",
      "sensu.io/managed_by": "sensuctl"
    },
    "created_by": "admin"
  }
]
```

response codes

- **Success**: 201 (Created)

- **Malformed**: 400 (Bad Request)

- **Error**: 500 (Internal Server Error)

Role bindings API

Get all role bindings

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "subjects": [
      {
        "type": "Group",
        "name": "readers"
      }
    ],
    "role_ref": {
      "type": "Role",
      "name": "read-only"
    },
    "metadata": {
      "name": "readers-group-binding",
```

```
        "namespace": "default",
        "created_by": "admin"
      }
    }
  ]
```

API Specification

/rolebindings (GET)	
description	Returns the list of role bindings.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	```[
{
"subjects": [
{
"type": "Group",
"name": "readers"
}
],
"role_ref": {
"type": "Role",
"name": "read-only"
},
"metadata": {
``

```
            "name": "readers-group-binding",
            "namespace": "default",
            "created_by": "admin"
        }
    }
]
```

Create a new role binding

The `/rolebindings` API endpoint provides HTTP POST access to create Sensu role bindings.

Example

In the following example, an HTTP POST request is submitted to the `/rolebindings` API endpoint to create a role binding named `readers-group-binding`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings
```

```
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	

```
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default"
  }
}
```

response codes

- ⌐ **Success**: 201 (Created)
- ⌐ **Malformed**: 400 (Bad Request)
- ⌐ **Error**: 500 (Internal Server Error)

Get a specific role binding

The `/rolebindings/:rolebinding` API endpoint provides HTTP GET access to role binding data for specific `:rolebinding` definitions, by role binding `name`.

Example

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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

API Specification

/rolebindings/:rolebinding (GET)

description	Returns the specified role binding.
example url	http://hostname:8080/api/core/v2/namespaces/default/role bindings/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",
    "created_by": "admin"
  }
}
```

Create or update a role binding

The `/rolebindings/:rolebinding` API endpoint provides HTTP PUT access to create or update role binding data for specific `:rolebinding` definitions, by role binding `name`.

Example

In the following example, an HTTP PUT request is submitted to the `/rolebindings/:rolebinding` API endpoint to create the role binding `dev-binding`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "devs"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "workflow-creator"
  },
  "metadata": {
    "name": "dev-binding",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding

HTTP/1.1 201 Created
```

API Specification

/rolebindings/:rolebinding (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)

Update a role binding with PATCH

The `/rolebindings/:rolebinding` API endpoint provides HTTP PATCH access to update `:rolebinding` definitions, specified by role binding name.

> **NOTE**: *You cannot change a resource's* `name` *or* `namespace` *with a PATCH request. Use a* *PUT request* *instead.*
>
> *Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.*

Example

In the following example, an HTTP PATCH request is submitted to the `/rolebindings/:rolebinding` API endpoint to add a group to the subjects array for the `dev-binding` role binding, resulting in an HTTP `200 OK` response and the updated role binding definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "dev_team_1"
    },
    {
      "type": "Group",
      "name": "dev_team_2"
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/rolebindings/dev-binding


HTTP/1.1 200 OK
```

API Specification

/rolebindings/:rolebinding (PATCH)	
description	Updates the specified Sensu role binding.
example URL	http://hostname:8080/api/core/v2/rolebindings/dev-binding

payload	`{`
	`  "subjects": [`
	`    {`

```
            "type": "Group",
            "name": "dev_team_1"
          },
          {
            "type": "Group",
            "name": "dev_team_2"
          }
        ]
      }
```

response codes

- ¬ **Success**: 200 (OK)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Delete a role binding

The `/rolebindings/:rolebinding` API endpoint provides HTTP DELETE access to delete a role binding from Sensu (specified by the role binding name).

Example

The following example shows a request to the `/rolebindings/:rolebinding` API endpoint to delete the role binding `dev-binding`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding \
-H "Authorization: Key $SENSU_API_KEY"


HTTP/1.1 204 No Content
```

API Specification

/rolebindings/:rolebinding (DELETE)

description	Removes the specified role binding from Sensu.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/dev-binding
response codes	⌐ **Success**: 204 (No Content) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

Roles API

Get all roles

The `/roles` API endpoint provides HTTP GET access to <u>role</u> data.

Example

The following example demonstrates a request to the `/roles` API endpoint, resulting in a JSON array that contains <u>role definitions</u>.

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "events"
        ],
        "resource_names": null
      }
    ],
```

```
    "metadata": {
      "name": "event-reader",
      "namespace": "default",
      :created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "read-only",
      "namespace": "default",
      "created_by": "admin"
    }
  }
]
```

API Specification

/roles (GET)	
description	Returns the list of roles.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response filtering	This endpoint supports API response filtering.
response type	Array

response codes	
	⌐ **Success**: 200 (OK)
	⌐ **Error**: 500 (Internal Server Error)

output	

```
[
  {
    "rules": [
      {
        "verbs": [
          "get",
          "list"
        ],
        "resources": [
          "events"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "event-reader",
      "namespace": "default",
      "created_by": "admin"
    }
  },
  {
    "rules": [
      {
        "verbs": [
          "get"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "read-only",
      "namespace": "default",
      "created by": "admin"
```

```
    }
  }
]
```

Create a new role

The `/roles` API endpoint provides HTTP POST access to create Sensu roles.

Example

In the following example, an HTTP POST request is submitted to the `/roles` API endpoint to create a role named `event-reader`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": []
    }
  ],
  "metadata": {
    "name": "event-reader",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles
```

```
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)

Get a specific role

The `/roles/:role` API endpoint provides HTTP GET access to <u>role data</u> 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` <u>definition</u> (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: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

API Specification

/roles/:role (GET)

description	Returns the specified Sensu role.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles/read-only
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	

```json
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default",
    "created_by": "admin"
  }
}
```

Create or update a role

The `/roles/:role` API endpoint provides HTTP PUT access to create or update specific `:role` definitions, by role name.

Example

In the following example, an HTTP PUT request is submitted to the `/roles/:role` API endpoint to create the role `read-only` . The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only

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	{

```
    "rules": [
      {
        "verbs": [
          "read"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "read-only",
      "namespace": "default"
    }
  }
```

---

response codes

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Update a role with PATCH

The `/roles/:role` API endpoint provides HTTP PATCH access to update `:role` definitions, specified by role name.

> **NOTE**: You cannot change a resource's `name` or `namespace` with a PATCH request. Use a _PUT request_ instead.
>
> Also, you cannot add elements to an array with a PATCH request — you must replace the entire array.

Example

In the following example, an HTTP PATCH request is submitted to the `/roles/:role` API endpoint to update the verbs array within the rules array for the `global-event-admin` role, resulting in an HTTP `200 OK` response and the updated role definition.

We support JSON merge patches, so you must set the `Content-Type` header to `application/merge-patch+json` for PATCH requests.

```
curl -X PATCH \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/merge-patch+json' \
-d '{
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
}' \
http://127.0.0.1:8080/api/core/v2/roles/event-reader


HTTP/1.1 200 OK
```

API Specification

/roles/:role (PATCH)	
description	Updates the specified Sensu role.
example URL	http://hostname:8080/api/core/v2/roles/event-reader
payload	{ "rules": [

```
        {
          "verbs": [
            "get",
            "list"
          ],
          "resources": [
            "events"
          ],
          "resource_names": null
        }
      ]
    }
```

---

response codes

- **Success**: 200 (OK)

- **Malformed**: 400 (Bad Request)

- **Error**: 500 (Internal Server Error)

Delete a role

The `/roles/:role` API endpoint provides HTTP DELETE access to delete a role from Sensu (specified by the role name).

Example

The following example shows a request to the `/roles/:role` API endpoint to delete the role `read-only`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only \
-H "Authorization: Key $SENSU_API_KEY"

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)

Searches API

NOTE: Requests to the searches API require you to authenticate with a Sensu _API key_ or _access token_. The code examples in this document use the _environment variable_ `$SENSU_API_KEY` to represent a valid API key in API requests.

Get all searches

The `/searches` API endpoint provides HTTP GET access to the list of saved searches.

Example

The following example demonstrates a request to the `/search` API endpoint, resulting in a JSON array that contains saved search definitions.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "incidents-us-west",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
```

```
          "labelSelector:region == \"us-west-1\"",
          "status:incident"
        ],
        "resource": "core.v2/Event"
      }
    },
    {
      "type": "Search",
      "api_version": "searches/v1",
      "metadata": {
        "name": "silenced-events",
        "namespace": "default"
      },
      "spec": {
        "parameters": [
          "silenced:true"
        ],
        "resource": "core.v2/Event"
      }
    },
    {
      "type": "Search",
      "api_version": "searches/v1",
      "metadata": {
        "name": "web-agent",
        "namespace": "default"
      },
      "spec": {
        "parameters": [
          "class:agent",
          "subscription:web"
        ],
        "resource": "core.v2/Entity"
      }
    }
  ]
```

API Specification

/searches (GET)	
description	Returns the list of saved searches.
example url	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)
output	

```json
[
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "incidents-us-west",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "labelSelector:region == \"us-west-1\"",
        "status:incident"
      ],
      "resource": "core.v2/Event"
    }
  },
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "silenced-events",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "silenced:true"
```

```
      ],
      "resource": "core.v2/Event"
    }
  },
  {
    "type": "Search",
    "api_version": "searches/v1",
    "metadata": {
      "name": "web-agent",
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "class:agent",
        "subscription:web"
      ],
      "resource": "core.v2/Entity"
    }
  }
]
```

Get a specific search

The `/searches/:search` API endpoint provides HTTP GET access to a specific `:search` definition, by the saved search `name`.

Example

In the following example, querying the `/searches/:search` API endpoint returns a JSON map that contains the requested `:search` definition (in this example, for the `:search` named `silenced-events`).

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches/silence
d-events \
-H "Authorization: Key $SENSU_API_KEY"
```

```
HTTP/1.1 200 OK
{
  "type": "Search",
  "api_version": "searches/v1",
  "metadata": {
    "name": "silenced-events",
    "namespace": "default"
  },
  "spec": {
    "parameters": [
      "silenced:true"
    ],
    "resource": "core.v2/Event"
  }
}
```

API Specification

/searches/:search (GET)	
description	Returns the specified search.
example url	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
response type	Map
response codes	⌐ **Success**: 200 (OK) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)
output	```{` ` "type": "Search",` ` "api_version": "searches/v1",` ` "metadata": {` ` "name": "silenced-events",```

```json
      "namespace": "default"
    },
    "spec": {
      "parameters": [
        "silenced:true"
      ],
      "resource": "core.v2/Event"
    }
  }
```

Create or update a search

The `/searches/:search` API endpoint provides HTTP PUT access to create or update a saved search by the saved search `name`.

Example

In the following example, an HTTP PUT request is submitted to the `/searches/:search` API endpoint to create or update a saved search for events that are silenced. The request includes the saved search definition in the request body and returns a successful HTTP `200 OK` response and the created or updated saved search definition.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "Search",
  "api_version": "searches/v1",
  "metadata": {
    "name": "silenced-events",
    "namespace": "default"
  },
  "spec": {
    "parameters": [
      "silenced:true"
    ],
    "resource": "core.v2/Event"
```

```
    }
}' \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches/silence
d-events


HTTP/1.1 200 OK
```

API Specification

/searches/:search (PUT)

description	Creates or updates the specified saved search.
example URL	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events

payload	<pre>{
	  "type": "Search",
	  "api_version": "searches/v1",
	  "metadata": {
	    "name": "silenced-events",
	    "namespace": "default"
	  },
	  "spec": {
	    "parameters": [
	      "silenced:true"
	    ],
	    "resource": "core.v2/Event"
	  }
	}</pre>

response codes	⌐ **Success**: 201 (Created) ⌐ **Malformed**: 400 (Bad Request) ⌐ **Error**: 500 (Internal Server Error)

Delete a search

The `/searches/:search` API endpoint provides HTTP DELETE access to delete a saved search from Sensu (specified by the saved search name).

Example

The following example shows a request to the `/searches/:search` API endpoint to delete the saved search `silenced-events`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/searches/v1/namespaces/default/searches/silence
d-events

HTTP/1.1 204 No Content
```

API Specification

/searches/:search (DELETE)	
description	Removes a saved search from Sensu (specified by the search name).
example url	http://hostname:8080/api/enterprise/searches/v1/namespaces/default/searches/silenced-events
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Secrets API

NOTE: Requests to the secrets API require you to authenticate with a Sensu API key or access token. The code examples in this document use the environment variable `$SENSU_API_KEY` *to represent a valid API key in API requests.*

Get all secrets providers

The `/providers` API endpoint provides HTTP GET access to a list of secrets providers.

Example

The following example demonstrates a request to the `/providers` API endpoint, resulting in a list of secrets providers.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers \
-H "Authorization: Key $SENSU_API_KEY"
[
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "my_vault",
      "created_by": "admin"
    },
    "spec": {
      "client": {
        "address": "https://vaultserver.example.com:8200",
        "token": "VAULT_TOKEN",
```

```
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
]
```

> **NOTE**: In addition to the `VaultProvider` type, the secrets API also includes a built-in `Env` secrets provider type that can retrieve backend _environment variables_ as secrets. Learn more in the _secrets providers reference_.

API Specification

/providers (GET)

description	Returns the list of secrets providers.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers
query parameters	`types` : Defines which type of secrets provider to retrieve. Join with `&` to retrieve multiple types: `?types=Env&types=VaultProvider` .
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)

output

```
[
  {
    "type": "VaultProvider",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "my_vault",
      "created_by": "admin"
    },
    "spec": {
      "client": {
        "address": "https://vaultserver.example.com:8200",
        "token": "VAULT_TOKEN",
        "version": "v1",
        "tls": {
          "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
        },
        "max_retries": 2,
        "timeout": "20s",
        "rate_limiter": {
          "limit": 10.0,
          "burst": 100
        }
      }
    }
  }
]
```

Get a specific secrets provider

The `/providers/:provider` API endpoint provides HTTP GET access to data for a specific secrets `:provider`, by provider name.

Example

In the following example, querying the `/providers/:provider` API endpoint returns a JSON map that contains the requested `:provider`, `my_vault`.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers/my_vault \
-H "Authorization: Key $SENSU_API_KEY"
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "my_vault",
    "created_by": "admin"
  },
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
```

API Specification

/providers/:provider (GET)	
description	Returns the specified secrets provider.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers/my_vault
response type	Map

response codes

- ⌐ **Success**: 200 (OK)

- ⌐ **Missing**: 404 (Not Found)

- ⌐ **Error**: 500 (Internal Server Error)

output

```json
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "my_vault",
    "created_by": "admin"
  },
  "spec": {
    "client": {
      "address":
"https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert":
"/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
```

Create or update a secrets provider

The `/providers/:provider` API endpoint provides HTTP PUT access to create or update a specific `:provider` , by provider name.

Example

The following example demonstrates a request to the `/providers/:provider` API endpoint to update the provider `my_vault`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "my_vault"
  },
  "spec": {
    "client": {
      "address": "https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert": "/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}' \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers/my_vault

HTTP/1.1 200 OK
```

API Specification

/providers/:provider (PUT)

description	Creates or updates the specified secrets provider. The provider resource and API version cannot be altered.
example URL	http://hostname:8080/api/enterprise/secrets/v1/providers/my_vault
payload	

```json
{
  "type": "VaultProvider",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "my_vault"
  },
  "spec": {
    "client": {
      "address":
"https://vaultserver.example.com:8200",
      "token": "VAULT_TOKEN",
      "version": "v1",
      "tls": {
        "ca_cert":
"/etc/ssl/certs/vault_ca_cert.pem"
      },
      "max_retries": 2,
      "timeout": "20s",
      "rate_limiter": {
        "limit": 10.0,
        "burst": 100
      }
    }
  }
}
```

response codes	

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Delete a secrets provider

The `/providers/:provider` API endpoint provides HTTP DELETE access to delete the specified provider from Sensu.

Example

The following example shows a request to the `/providers/:provider` API endpoint to delete the provider `my_vault`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/secrets/v1/providers/my_vault


HTTP/1.1 204 No Content
```

API Specification

/providers/:provider (DELETE)	
description	Deletes the specified provider from Sensu.
example url	http://hostname:8080/api/enterprise/secrets/v1/providers/my_vault
response codes	⌐ **Success**: 204 (No Content) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

Get all secrets

The `/secrets` API endpoint provides HTTP GET access to a list of secrets.

Example

The following example demonstrates a request to the `/secrets` API endpoint, resulting in a list of secrets for the specified namespace.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
[
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sensu-ansible-token",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/ansible#token",
      "provider": "ansible_vault"
    }
  }
]
```

API Specification

/secrets (GET)	
description	Returns the list of secrets for the specified namespace.
example url	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets
response filtering	This endpoint supports API response filtering.

response type	Array
response codes	
	⌐ **Success**: 200 (OK)
	⌐ **Error**: 500 (Internal Server Error)
output	

```json
[
  {
    "type": "Secret",
    "api_version": "secrets/v1",
    "metadata": {
      "name": "sensu-ansible-token",
      "namespace": "default",
      "created_by": "admin"
    },
    "spec": {
      "id": "secret/ansible#token",
      "provider": "ansible_vault"
    }
  }
]
```

Get a specific secret

The `/secrets/:secret` API endpoint provides HTTP GET access to data for a specific `secret`, by secret name.

Example

In the following example, querying the `/secrets/:secret` API endpoint returns a JSON map that contains the requested `:secret`.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-
```

```
ansible-token \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default",
    "created_by": "admin"
  },
  "spec": {
    "id": "secret/ansible#token",
    "provider": "ansible_vault"
  }
}
```

API Specification

/secrets/:secret (GET)	
description	Returns the specified secret.
example url	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token
response type	Map
response codes	⌐ **Success**: 200 (OK) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)
output	```{
"type": "Secret",
"api_version": "secrets/v1",
"metadata": {```

```
        "name": "sensu-ansible-token",
        "namespace": "default",
        "created_by": "admin"
      },
      "spec": {
        "id": "secret/ansible#token",
        "provider": "ansible_vault"
      }
    }
```

Create or update a secret

The `/secrets/:secret` API endpoint provides HTTP PUT access to create or update a specific `secret`, by secret name.

Example

The following example demonstrates a request to the `/secrets/:secret` API endpoint to update the secret `sensu-ansible-token`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "Secret",
  "api_version": "secrets/v1",
  "metadata": {
    "name": "sensu-ansible-token",
    "namespace": "default"
  },
  "spec": {
    "id": "secret/ansible#token",
    "provider": "ansible_vault"
  }
}' \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-
ansible-token
```

```
HTTP/1.1 200 OK
```

API Specification

/secrets/:secret (PUT)	
description	Creates or updates the specified secret.
example URL	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token

payload	
	``json
	{
	  "type": "Secret",
	  "api_version": "secrets/v1",
	  "metadata": {
	    "name": "sensu-ansible-token",
	    "namespace": "default"
	  },
	  "spec": {
	    "id": "secret/ansible#token",
	    "provider": "ansible_vault"
	  }
	}
	``

response codes

- ⌐ **Success**: 201 (Created)

- ⌐ **Malformed**: 400 (Bad Request)

- ⌐ **Error**: 500 (Internal Server Error)

Delete a secret

The `/secrets/:secret` API endpoint provides HTTP DELETE access to delete the specified secret from Sensu.

Example

The following example shows a request to the `/secrets/:secret` API endpoint to delete the secret `sensu-ansible-token`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-
ansible-token

HTTP/1.1 204 No Content
```

API Specification

/secrets/:secret (DELETE)	
description	Deletes the specified secret from Sensu.
example url	http://hostname:8080/api/enterprise/secrets/v1/namespaces/default/secrets/sensu-ansible-token
response codes	⌐ **Success**: 204 (No Content) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

Silencing API

Get all silences

The `/silenced` API endpoint provides HTTP GET access to <u>silencing entry</u> data.

Example

The following example demonstrates a request to the `/silenced` API endpoint, resulting in a JSON array that contains <u>silencing entry definitions</u>.

```
curl -X GET \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "*:http",
      "namespace": "default",
      "created_by": "admin"
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "check": "http",
    "reason": "Testing",
    "begin": 1605024595,
    "expire_at": 0
```

```
    },
    {
      "metadata": {
        "name": "linux:*",
        "namespace": "default",
        "created_by": "admin"
      },
      "expire": -1,
      "expire_on_resolve": false,
      "creator": "admin",
      "reason": "reason for silence",
      "subscription": "linux",
      "begin": 1542671205,
      "expire_at": 0
    }
  ]
```

API Specification

/silenced (GET)	
description	Returns the list of silences.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced
pagination	This endpoint does not support pagination.
response filtering	This endpoint supports API response filtering.
response type	Array
response codes	⌐ **Success**: 200 (OK) ⌐ **Error**: 500 (Internal Server Error)

output	```[
	  {
	    "metadata": {```

```
        "name": "*:http",
        "namespace": "default",
        "created_by": "admin"
      },
      "expire": -1,
      "expire_on_resolve": false,
      "creator": "admin",
      "check": "http",
      "reason": "Testing",
      "begin": 1605024595,
      "expire_at": 0
    },
    {
      "metadata": {
        "name": "linux:*",
        "namespace": "default",
        "created_by": "admin"
      },
      "expire": -1,
      "expire_on_resolve": false,
      "creator": "admin",
      "reason": "reason for silence",
      "subscription": "linux",
      "begin": 1542671205,
      "expire_at": 0
    }
  ]
```

Create a new silence

The `/silenced` API endpoint provides HTTP POST access to create silencing entries.

Example

In the following example, an HTTP POST request is submitted to the `/silenced` API endpoint to create the silencing entry `linux:check-cpu`. The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "linux:check-cpu",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced


HTTP/1.1 201 Created
```

Here's another example that shows an HTTP POST request to the `/silenced` API endpoint to create the silencing entry `*:http`, which will create a silence for any event with the check name `http`, regardless of the originating entities' subscriptions.The request returns a successful HTTP `201 Created` response.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "*:http",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
```

```
  "check": "http",
  "reason": "Testing"
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced


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	
	``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
	}
	``

response codes

- ¬ **Success**: 201 (Created)

- ¬ **Malformed**: 400 (Bad Request)

- ¬ **Error**: 500 (Internal Server Error)

Get a specific silence

The `/silenced/:silenced` API endpoint provides HTTP GET access to silencing entry data for specific `:silenced` definitions, by silencing entry name.

Example

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: Key $SENSU_API_KEY" \
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",
    "created_by": "admin",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}
```

API Specification

/silenced/:silenced (GET)

description	Returns the specified silencing entry.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu
response type	Map
response codes	⌐ **Success**: 200 (OK) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

output	
	``json
	{
	  "metadata": {
	    "name": "linux:check-cpu",
	    "namespace": "default",
	    "created_by": "admin",
	    "labels": null,
	    "annotations": null
	  },
	  "expire": -1,
	  "expire_on_resolve": false,
	  "creator": "admin",
	  "reason": "reason for silence",
	  "subscription": "linux",
	  "begin": 1542671205
	}
	``

Create or update a silence

The `/silenced/:silenced` API endpoint provides HTTP PUT access to create or update specific `:silenced` definitions, by silencing entry name.

Example

In the following example, an HTTP PUT request is submitted to the `/silenced/:silenced` API

endpoint to create the silencing entry `linux:check-server`. The request returns a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "metadata": {
    "name": "linux:check-server",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "reason": "reason for silence",
  "subscription": "linux",
  "begin": 1542671205
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-server

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)

Delete a silence

The `/silenced/:silenced` API endpoint provides HTTP DELETE access to delete a silencing entry (specified by the silencing entry name).

Example

In the following example, querying the `/silenced/:silenced` API endpoint to delete the the silencing entry named `linux:check-cpu` results in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu


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)

Get all silences for a specific subscription

The `/silenced/subscriptions/:subscription` API endpoint provides HTTP GET access to silencing entry data by subscription name.

Example

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: Key $SENSU_API_KEY" \
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",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
```

```
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```

API Specification

/silenced /subscriptions /:subscription (GET)	
description	Returns all silences for the specified subscription.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/subscriptions/linux
pagination	This endpoint supports pagination using the `limit` and `continue` query parameters.
response type	Array
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

output	<pre>[
	  {
	    "metadata": {
	      "name": "linux:check-cpu",
	      "namespace": "default",
	      "created_by": "admin",
	      "labels": null,
	      "annotations": null
	    },</pre>

```
        "expire": -1,
        "expire_on_resolve": false,
        "creator": "admin",
        "reason": "reason for silence",
        "subscription": "linux",
        "begin": 1542671205
    }
]
```

Get all silences for a specific check

The `/silenced/checks/:check` API endpoint provides HTTP GET access to silencing entry data by check name.

Example

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: Key $SENSU_API_KEY" \
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",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
```

```
    "check": "linux",
    "begin": 1542671205
  }
]
```

API Specification

<table>
<tr><td colspan="2">/silenced/checks /:check (GET)</td></tr>
<tr><td>description</td><td>Returns all silences for the specified check.</td></tr>
<tr><td>example url</td><td>http://hostname:8080/api/core/v2/namespaces/default/silenced/checks/check-cpu</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 type</td><td>Array</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>

```
[
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
      "created_by": "admin",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": "reason for silence",
```
</td></tr>
</table>

```
        "check": "linux",
        "begin": 1542671205
    }
]
```

Tessen API

The Tessen API provides HTTP access to manage [Tessen](#) configuration. Access to the Tessen API is restricted to the default `admin` user.

Get the active Tessen configuration

The `/tessen` API endpoint provides HTTP GET access to the active Tessen configuration.

Example

The following example demonstrates an HTTP GET request to the `/tessen` API endpoint. 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: Key $SENSU_API_KEY"

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	```json
	{
	  "opt_out": false
	}
	``

Opt in to or out of Tessen

The `/tessen` API endpoint provides HTTP PUT access to opt in to or opt out of Tessen for unlicensed Sensu instances.

> **NOTE**: Tessen is enabled by default on Sensu backends and required for _licensed_ Sensu instances. If you have a licensed instance and want to opt out of Tessen, contact your account manager.

Example

In the following example, an HTTP PUT request is submitted to the `/tessen` API endpoint to opt in to Tessen using the `opt_out` attribute. The request returns an HTTP `200 OK` response and the resulting Tessen configuration.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
```

```
-H 'Content-Type: application/json' \
-d '{
  "opt_out": false
}' \
http://127.0.0.1:8080/api/core/v2/tessen


HTTP/1.1 200 OK
{
  "opt_out": false
}
```

API Specification

/tessen (PUT)	
description	Updates the active Tessen configuration for unlicensed Sensu instances. Tessen is enabled by default on Sensu backends and required for licensed Sensu instances.
example url	http://hostname:8080/api/core/v2/tessen
request parameters	Required: `opt_out` (for unlicensed instances, set to `false` to enable Tessen; set to `true` to opt out of Tessen).
response codes	⌐ **Success**: 200 (OK) ⌐ **Missing**: 404 (Not Found) ⌐ **Error**: 500 (Internal Server Error)

example output	```
	{
	  "opt_out": false
	}
	``

Users API

Get all users

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: Key $SENSU_API_KEY"

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
  }
``` |

```
    ]
```

Create a new user

The `/users` API endpoint provides HTTP POST access to create a user using Sensu's basic authentication provider.

Example

The following example demonstrates a POST request to the `/users` API endpoint to create the user `alice`, resulting in an HTTP `201 Created` response and the created user definition.

```
curl -X POST \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "temporary",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users


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

```
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "temporary",
  "disabled": false
}
```

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

Get a specific user

The `/users/:user` API endpoint provides HTTP GET access to user data for a specific user by `username`.

Example

In the following example, querying the `/users/:user` API returns a JSON map that contains the requested `:user` definition (in this example, for the `alice` user).

```
curl -X GET \
http://127.0.0.1:8080/api/core/v2/users/alice \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
```

```json
{
  "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\n{\n "username": "alice",\n "groups": [\n "ops"\n],\n "disabled": false\n}\n```

Create or update a user

The `/users/:user` API endpoint provides HTTP PUT access to create or update <u>user data</u> for a

specific user by `username` .

> **NOTE**: Use the `PUT /users/:user/reset_password` or `PUT /users/:user/password` API
> endpoints to reset or change the user password, respectively.

Example

The following example demonstrates a PUT request to the `/users` API endpoint to update the user
`alice` (for example, to add the user to the `devel` group), resulting in an HTTP `201 Created`
response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops",
    "devel"
  ],
  "password": "password",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users/alice

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		
	{	

```json
    "username": "alice",
    "groups": [
      "ops",
      "devel"
    ],
    "password": "password",
    "disabled": false
  }
```

response codes

⌐ **Success**: 201 (Created)

⌐ **Malformed**: 400 (Bad Request)

⌐ **Error**: 500 (Internal Server Error)

Delete a user

The `/users/:user` API endpoint provides HTTP DELETE access to disable a specific user by `username`.

Example

In the following example, an HTTP DELETE request is submitted to the `/users/:user` API endpoint to disable the user `alice`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
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 _reinstate_ disabled users.

API Specification

/users/:user (DELETE)	
description	Disables the specified user.
example url	http://hostname:8080/api/core/v2/users/alice
response codes	¬ **Success**: 204 (No Content) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)

Reset a user's password

The `/users/:user/reset_password` API endpoint provides HTTP PUT access to reset a user's password.

> **NOTE**: The `/users/:user/reset_password` API endpoint requires explicit `users` _permissions_. With these permissions, you can use `/users/:user/reset_password` to reset a user's password. This differs from the `/users/:user/password` API endpoint, which allows users to change their own passwords without explicit permissions.

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/reset_password` API endpoint to reset the password for the user `alice`, resulting in an HTTP `201 Created` response.

The `password_hash` is the user's new password, hashed via bcrypt.Use `sensuctl user hash-password` to generate the `password_hash`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
```

```
-d '{
  "username": "alice",
  "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm"
}' \
http://127.0.0.1:8080/api/core/v2/users/alice/reset_password


HTTP/1.1 201 Created
```

API Specification

/users/:user/reset_password (PUT)

description	Resets the password for the specified Sensu user.
example URL	http://hostname:8080/api/core/v2/users/alice/reset_password

payload parameters	Required:
	
	  ¬ `username` : string; the username for the Sensu user
	
	  ¬ `password_hash` : string; the new user password, hashed via bcrypt

payload	<pre>{
	  "username": "alice",
	  "password_hash":
	"$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLG
	HjEqxJbzPsreQAyNm"
	}</pre>

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Change your password

The `/users/:user/password` API endpoint provides HTTP PUT access to change your Sensu user password.

> **NOTE**: The `/users/:user/password` API endpoint allows a user to update their own password, without any permissions. This differs from the `/users/:user/reset_password` API endpoint, which requires explicit `users` permissions to change the user password.

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/password` API endpoint to update the password for the user `alice`, resulting in an HTTP `201 Created` response.

The `password` is your current password in cleartext. The `password_hash` is your new password hashed via bcrypt. Use `sensuctl user hash-password` to generate the `password_hash`.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "password": "P@ssw0rd!",
  "password_hash": "$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJbzPsreQAyNm"
}' \
http://127.0.0.1:8080/api/core/v2/users/alice/password

HTTP/1.1 201 Created
```

API Specification

/users/:user/password (PUT)

description	Changes the password for the specified Sensu user.
example URL	http://hostname:8080/api/core/v2/users/alice/password

payload parameters	Required:
	
	    ¬ `username` : string; the username for the Sensu user
	
	    ¬ `password` : string; the user's current password in cleartext
	
	    ¬ `password_hash` : string; the user's hashed password via <u>bcrypt</u>

payload	<code>
	{
	  "username": "alice",
	  "password": "P@ssw0rd!",
	  "password_hash":
	"$5f$14$.brXRviMZpbaleSq9kjoUuwm67V/s4IziOLGHjEqxJ
	bzPsreQAyNm"
	}
	</code>

response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Reinstate a disabled user

The `/users/:user/reinstate` API endpoint provides HTTP PUT access to reinstate a disabled user.

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/reinstate` API endpoint to reinstate the disabled user `alice` , resulting in an HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
http://127.0.0.1:8080/api/core/v2/users/alice/reinstate


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)

Remove a user from all groups

The `/users/:user/groups` API endpoint provides HTTP DELETE access to remove the specified user from all groups.

Example

In the following example, an HTTP DELETE request is submitted to the `/users/:user/groups` API endpoint to remove the user `alice` from all groups within Sensu, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
```

```
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups


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)

Assign a user to a group

The `/users/:user/groups/:group` API endpoint provides HTTP PUT access to assign a user to a group.

Example

In the following example, an HTTP PUT request is submitted to the `/users/:user/groups/:group` API endpoint to add the user `alice` to the group `ops`, resulting in a successful HTTP `201 Created` response.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops

HTTP/1.1 201 Created
```

API Specification

/users/:user/groups/:group (PUT)	
description	Adds the specified user to the specified group.
example URL	http://hostname:8080/api/core/v2/users/alice/groups/ops
response codes	¬ **Success**: 201 (Created) ¬ **Malformed**: 400 (Bad Request) ¬ **Error**: 500 (Internal Server Error)

Remove a user from a specific group

The `/users/:user/groups/:group` API endpoint provides HTTP DELETE access to remove the specified user from a specific group.

Example

In the following example, an HTTP DELETE request is submitted to the `/users/:user/groups/:group` API endpoint to remove the user `alice` from the group `ops`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops

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

Get the Sensu backend and etcd versions

The `/version` API endpoint provides HTTP GET access to the Sensu backend and etcd versions for the Sensu instance.

Example

The following example demonstrates a 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": "3.3.22",
    "etcdcluster": "3.3.0"
  },
  "sensu_backend": "6.0.0"
}
```

API Specification

/version (GET)	
description	Returns the etcd server version and Sensu backend version. For clustered Sensu installations with the default embedded etcd, also returns the etcd cluster version (which may not match the etcd server version or the cluster versions of other backends in the cluster).

example url	http://hostname:8080/version
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)

output	<pre>{
	  "etcd": {
	    "etcdserver": "3.3.22",
	    "etcdcluster": "3.3.0"
	  },
	  "sensu_backend": "6.x.x"
	}</pre>

Web UI configuration API

NOTE: Requests to the web UI configuration API require you to authenticate with a Sensu *API key* or *access token*. The code examples in this document use the *environment variable* `$SENSU_API_KEY` to represent a valid API key in API requests.

Get the web UI configuration

The `/config` API endpoint provides HTTP GET access to the global web UI configuration.

Example

The following example demonstrates a request to the `/config` API endpoint, resulting in a JSON array that contains the global web UI configuration.

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/web/v1/config \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json'

HTTP/1.1 200 OK
[
  {
    "type": "GlobalConfig",
    "api_version": "web/v1",
    "metadata": {
      "name": "custom-web-ui",
      "created_by": "admin"
    },
    "spec": {
```

```
      "always_show_local_cluster": false,
      "default_preferences": {
        "page_size": 50,
        "theme": "sensu"
      },
      "link_policy": {
        "allow_list": true,
        "urls": [
          "https://example.com",
          "steamapp://34234234",
          "//google.com",
          "//*.google.com",
          "//bob.local",
          "https://grafana-host/render/metrics?
 width=500&height=250#sensu.io.graphic"
        ]
      }
    }
  }
]
```

API Specification

/web (GET)	
description	Returns the list of global web UI configurations.
example url	http://hostname:8080/api/enterprise/web/v1/config
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Error**: 500 (Internal Server Error)
output	```[
{
"type": "GlobalConfig",```

```
      "api_version": "web/v1",
      "metadata": {
        "name": "custom-web-ui",
        "created_by": "admin"
      },
      "spec": {
        "always_show_local_cluster": false,
        "default_preferences": {
          "page_size": 50,
          "theme": "sensu"
        },
        "link_policy": {
          "allow_list": true,
          "urls": [
            "https://example.com",
            "steamapp://34234234",
            "//google.com",
            "//*.google.com",
            "//bob.local",
            "https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
          ]
        }
      }
    }
  ]
```

Get a specific web UI configuration

The `/config/:globalconfig` API endpoint provides HTTP GET access to global web UI configuration data, specified by configuration name.

Example

In the following example, querying the `/config/:globalconfig` API endpoint returns a JSON map that contains the requested `:globalconfig` definition (in this example, for the `:globalconfig` named `custom-web-ui`).

```
curl -X GET \
http://127.0.0.1:8080/api/enterprise/web/v1/config/custom-web-ui \
-H "Authorization: Key $SENSU_API_KEY"

HTTP/1.1 200 OK
{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui",
    "created_by": "admin"
  },
  "spec": {
    "always_show_local_cluster": false,
    "default_preferences": {
      "page_size": 50,
      "theme": "sensu"
    },
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
      ]
    }
  }
}
```

API Specification

/config/:globalconfig (GET)	
description	Returns the specified global web UI configuration.
example url	http://hostname:8080/api/enterprise/web/v1/config/custom-web-

ui	
response type	Map
response codes	¬ **Success**: 200 (OK) ¬ **Missing**: 404 (Not Found) ¬ **Error**: 500 (Internal Server Error)
output	<pre>{
"type": "GlobalConfig",
"api_version": "web/v1",
"metadata": {
"name": "custom-web-ui",
"created_by": "admin"
},
"spec": {
"always_show_local_cluster": false,
"default_preferences": {
"page_size": 50,
"theme": "sensu"
},
"link_policy": {
"allow_list": true,
"urls": [
"https://example.com",
"steamapp://34234234",
"//google.com",
"//*.google.com",
"//bob.local",
"https://grafana-host/render/metrics?
width=500&height=250#sensu.io.graphic"
]
}
}
}</pre>

Create and update a web UI configuration

The `/config/:globalconfig` API endpoint provides HTTP PUT access to create and update global web UI configurations, specified by configuration name.

Example

In the following example, an HTTP PUT request is submitted to the `/config/:globalconfig` API endpoint to update the `custom-web-ui` configuration, resulting in an HTTP `200 OK` response and the updated configuration definition.

```
curl -X PUT \
-H "Authorization: Key $SENSU_API_KEY" \
-H 'Content-Type: application/json' \
-d '{
  "type": "GlobalConfig",
  "api_version": "web/v1",
  "metadata": {
    "name": "custom-web-ui",
    "created_by": "admin"
  },
  "spec": {
    "always_show_local_cluster": false,
    "default_preferences": {
      "page_size": 50,
      "theme": "sensu"
    },
    "link_policy": {
      "allow_list": true,
      "urls": [
        "https://example.com",
        "steamapp://34234234",
        "//google.com",
        "//*.google.com",
        "//bob.local",
        "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
      ]
    }
  }
}' \
```

```
http://127.0.0.1:8080/api/enterprise/web/v1/config/custom-web-ui
```

```
HTTP/1.1 201 Created
```

API Specification

/config/:globalconfig (PUT)	
description	Creates or updates the specified global web UI configuration.
example URL	http://hostname:8080/api/enterprise/web/v1/config/custom-web-ui

payload	<code>{
	  "type": "GlobalConfig",
	  "api_version": "web/v1",
	  "metadata": {
	    "name": "custom-web-ui",
	    "created_by": "admin"
	  },
	  "spec": {
	    "always_show_local_cluster": false,
	    "default_preferences": {
	      "page_size": 50,
	      "theme": "sensu"
	    },
	    "link_policy": {
	      "allow_list": true,
	      "urls": [
	        "https://example.com",
	        "steamapp://34234234",
	        "//google.com",
	        "//*.google.com",
	        "//bob.local",
	        "https://grafana-host/render/metrics?width=500&height=250#sensu.io.graphic"
	      ]
	    }
	  }
	}</code>

```
      }
```

---

response codes

- ⌐ **Success**: 201 (Created)

- ⌐ **Malformed**: 400 (Bad Request)

- ⌐ **Error**: 500 (Internal Server Error)

Delete a web UI configuration

The `/config/:globalconfig` API endpoint provides HTTP DELETE access to delete a global web UI configuration from Sensu, specified by the configuration name.

Example

The following example shows a request to the `/config/:globalconfig` API endpoint to delete the global web UI configuration named `custom-web-ui`, resulting in a successful HTTP `204 No Content` response.

```
curl -X DELETE \
-H "Authorization: Key $SENSU_API_KEY" \
http://127.0.0.1:8080/api/enterprise/web/v1/config/custom-web-ui

HTTP/1.1 204 No Content
```

API Specification

/config/:globalconfig (DELETE)	
description	Removes the specified global web UI configuration from Sensu.
example url	http://hostname:8080/api/enterprise/web/v1/config/custom-web-ui

response codes

- ⌐ **Success**: 204 (No Content)
- ⌐ **Missing**: 404 (Not Found)
- ⌐ **Error**: 500 (Internal Server Error)

Reference Index

This index links to every reference in the Sensu documentation.Reference documentation includes specifications, examples, configuration notes, and other details about Sensu resources, the agent and backend, and Sensu query expressions.

- ¬ [Active Directory (AD)](#)

- ¬ [Agent](#)

- ¬ [API keys](#)

- ¬ [Assets](#)

- ¬ [Backend](#)

- ¬ [Checks](#)

- ¬ [Datastore](#)

- ¬ [Entities](#)

- ¬ [Etcd replicators](#)

- ¬ [Event filters](#)

- ¬ [Events](#)

- ¬ [Handlers](#)

- ¬ [Health](#)

- ¬ [Hooks](#)

- ¬ [License](#)

- ¬ [Lightweight Directory Access Protocol (LDAP)](#)

- ¬ [Metrics](#)

- ¬ [Mutators](#)

- ¬ [Namespaces](#)

- ¬ [OpenID Connect 1.0 protocol (OIDC)](#)

- ¬ [Plugins](#)

- ¬ [Role-based access control (RBAC)](#)

Plugins

Sensu plugins provide executable scripts or other programs that you can use as Sensu checks, handlers, and mutators.

Plugins are service-specific and have different setup and configuration requirements. Many Sensu-supported plugins include quick-start templates that you only need to edit to match your configuration. Each plugin has self-contained documentation with in-depth information about how to install and use it.

Find Sensu plugins

Search Bonsai, the Sensu asset hub, to find Sensu plugins. Bonsai lists hundreds of Sensu plugins with installation instructions and usage examples.

We also list popular Sensu-supported plugins in the supported integrations section.

Write your own custom plugins

Write your own Sensu plugins in almost any programming language with Sensu's plugin specification. The Sensu Go plugin SDK library provides a framework for building Sensu Go plugins so that all you need to do is define plugin arguments and input validation and execution functions.

If you are interested in sharing your plugin with other Sensu users, you can find guidance for contributing plugins, pinning versions, writing plugin READMEs, and transferring repos to community responsibility at the Sensu plugins community GitHub repo

Use Nagios plugins

The Sensu plugin specification is compatible with the Nagios plugin specification, so you can use Nagios plugins with Sensu without any modification. Sensu allows you to bring new life to the 50+ plugins in the official Nagios Plugins project, a mature source of monitoring plugins, and more than 4000 plugins in the Nagios Exchange.

Install Sensu plugins

Extend Sensu's functionality with plugins, which provide executables for performing status or metric checks, mutators for changing data to a desired format, and handlers for performing an action on a Sensu event.

Install plugins with dynamic runtime assets

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. To start using and deploying assets, read Use dynamic runtime assets to install plugins to become familiar with workflows that involve assets.

> **NOTE**: *Dynamic runtime assets are not required to use Sensu Go. You can install Sensu plugins using the sensu-install tool or a configuration management solution.*

Use Bonsai, the Sensu asset hub

Bonsai, the Sensu asset hub, is a centralized place for downloading and sharing dynamic runtime 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*

The `sensu-install` tool comes with an embedded version of Ruby, so you don't need to have Ruby installed on your system.

To install a Sensu Community plugin with Sensu Go:

1. Install the sensu-plugins-ruby package from packagecloud.

2. Run the `sensu-install` command to install plugins in the Sensu Community Plugins GitHub organization by repository name. Plugins are installed into `/opt/sensu-plugins-ruby/embedded/bin`.

To list all flags for the sensu-install command, run:

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

The response will be similar to this example:

```
Usage: sensu-install [options]
    -h, --help                       Display this message
    -v, --verbose                    Enable verbose logging
    -p, --plugin PLUGIN              Install a Sensu PLUGIN
    -P, --plugins PLUGIN[,PLUGIN]    PLUGIN or comma-delimited list of Sensu plugins
to install
    -e, --extension EXTENSION        Install a Sensu EXTENSION
    -E, --extensions EXTENSION[,EXT] EXTENSION or comma-delimited list of Sensu
extensions to install
    -s, --source SOURCE              Install Sensu plugins and extensions from a
custom SOURCE
    -c, --clean                      Clean up (remove) other installed versions of
the plugin(s) and/or extension(s)
    -x, --proxy PROXY                Install Sensu plugins and extensions via a
PROXY URL
```

For example, to install the Sensu InfluxDB plugin:

```
sudo sensu-install -p influxdb
```

To install a specific version of the Sensu InfluxDB plugin with `sensu-install` , run:

```
sudo sensu-install -p 'sensu-plugins-influxdb:2.0.0'
```

> **NOTE**: We recommend specifying the plugin version you want to install to maintain the stability of your observability infrastructure. If you do not specify a version to install, Sensu automatically installs the latest version, which may include breaking changes.

Use a configuration management tool or Sensu dynamic runtime assets to pin the versions of any plugins installed in production.

> **NOTE**: If a plugin is not Sensu Go-enabled and there is no analogue on Bonsai, you can add the necessary functionality to make the plugin compatible with Sensu Go. Follow the Discourse guide _Contributing Assets for Existing Ruby Sensu Plugins_ to walk through the process.

Troubleshoot the sensu-install tool

Some plugins require additional tools to install them successfully. An example is the Sensu disk checks plugin.

To download and update package information:

SHELL

```
sudo apt-get update
```

SHELL

```
sudo yum update
```

Depending on the plugin, you may need to install developer tool packages:

SHELL

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

SHELL

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

Use dynamic runtime assets to install plugins

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins. You can use assets to provide the plugins, libraries, and runtimes you need to automate your monitoring workflows. Read the asset reference for more information about dynamic runtime assets. This guide uses the Sensu PagerDuty Handler dynamic runtime asset as an example.

> **NOTE**: Dynamic runtime assets are not required to use Sensu Go. You can install Sensu plugins using the sensu-install tool or a configuration management solution.

Register the Sensu PagerDuty Handler asset

To add the Sensu PagerDuty Handler dynamic runtime asset to Sensu, use `sensuctl asset add <namespace/name>:<version>` :

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

The response should be similar to this example:

```
fetching bonsai asset: sensu/sensu-pagerduty-handler:1.2.0
added asset: sensu/sensu-pagerduty-handler:1.2.0

You have successfully added the Sensu asset resource, but the asset will not get
downloaded until
it's invoked by another Sensu resource (ex. check). To add this runtime asset to the
appropriate
resource, populate the "runtime_assets" field with ["pagerduty-handler"].
```

> **NOTE**: We recommend specifying the asset version you want to install to maintain the stability of

your observability infrastructure. If you do not specify a version to install, Sensu automatically installs the latest version, which may include breaking changes.

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

You can also open the **Release Assets** tab on asset pages in <u>Bonsai</u> to download the asset definition for your Sensu backend platform and architecture.

NOTE: Sensu does not download and install asset builds onto the system until they are needed for command execution. Read <u>the asset reference</u> for more information about asset builds.

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 <u>Sensu PagerDuty Handler</u> asset for Linux AMD64:

YML

```yml
---
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'
```

JSON

```json
{
  "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":
"5be236b5b9ccceb10920d3a171ada4ac4f4caaf87f822475cd48bd7f2fab3235fa298f79ef6f97b0eb6
498205740bb1af1120ca036fd3381edfebd9fb15aaa99",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ]
  }
}
```

Filters for *check* dynamic runtime assets should match entity platforms. Filters for *handler and filter* dynamic runtime assets should match your Sensu backend platform. If the provided filters are too restrictive for your platform, replace `os` and `arch` with any supported entity system attributes (for example, `entity.system.platform_family == 'rhel'`). You may also want to customize the asset `name` to reflect the supported platform (for example, `pagerduty-handler-linux`) and add custom attributes with `labels` and `annotations` .

Enterprise-tier dynamic runtime assets (like the ServiceNow and Jira event handlers) require a Sensu commercial license. For more information about commercial features and to activate your license, read Get started with commercial features.

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

SHELL

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

SHELL

```
sensuctl asset list --format wrapped-json
```

Create a workflow

With the asset downloaded and registered, you can use it in a monitoring workflow.Dynamic runtime assets may provide executable plugins intended for use with a Sensu check, handler, mutator, or hook, or JavaScript libraries intended to provide functionality for use in event filters.The details in Bonsai are the best resource for information about each asset's capabilities and configuration.

For example, to use the Sensu PagerDuty Handler asset, you would create a `pagerduty` handler that includes your PagerDuty service API key in place of `SECRET` and `pagerduty-handler` as a runtime asset:

YML

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

```json
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "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.yml` or `pagerduty-handler.json`), and add it to Sensu with sensuctl:

SHELL

```shell
sensuctl create --file pagerduty-handler.yml
```

SHELL

```shell
sensuctl create --file pagerduty-handler.json
```

Now that Sensu can create incidents in PagerDuty, you can automate this workflow by adding the

`pagerduty` handler to your Sensu service check definitions.Read Monitor server resources to learn more.

Next steps

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

- ¬ Assets reference

- ¬ Asset format specification

- ¬ Share assets on Bonsai

Follow Send PagerDuty alerts with Sensu to configure a check that generates status events and a handler that sends Sensu alerts to PagerDuty for non-OK events.

Assets reference

Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu <u>plugins</u>. You can use dynamic runtime assets to provide the plugins, libraries, and runtimes you need to automate your monitoring workflows. Sensu supports dynamic runtime assets for <u>checks</u>, <u>filters</u>, <u>mutators</u>, and <u>handlers</u>.

You can discover, download, and share dynamic runtime assets using <u>Bonsai, the Sensu asset hub</u>. Read <u>Use assets to install plugins</u> to get started.

> **NOTE**: Dynamic runtime assets are not required to use Sensu Go. You can install Sensu plugins using the <u>sensu-install</u> tool or a <u>configuration management</u> solution.

The Sensu backend executes handler, filter, and mutator dynamic runtime assets. The Sensu agent executes check dynamic runtime assets. At runtime, the backend or agent sequentially evaluates dynamic runtime assets that appear in the `runtime_assets` attribute of the handler, filter, mutator, or check being executed.

Dynamic runtime asset example (minimum required attributes)

This example shows a dynamic runtime asset resource definition that includes the minimum required attributes:

YML

```yml
---
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"
      }
    ]
  }
}
```

Dynamic runtime asset builds

A dynamic runtime asset build is the combination of an artifact URL, SHA512 checksum, and optional
Sensu query expression filters.Each asset definition may describe one or more builds.

> **NOTE**: Dynamic runtime assets that provide `url` and `sha512` attributes at the top level of the
> `spec` scope are single-build assets, and this form of asset defintion is deprecated.We recommend
> using multiple-build asset defintions, which specify one or more `builds` under the `spec` scope.

Asset example: Multiple builds

This example shows the resource definition for a dynamic runtime asset with multiple builds:

YML

```yaml
---
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:
487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a72b786ef0ca396a0a12c8d006
ac7fa21923e0e9ae63419a4d56aec41fccb574c1a5d3
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    headers:
      Authorization: 'Bearer {{ .annotations.asset_token | default "N/A" }}'
      X-Forwarded-For: client1, proxy1, proxy2
  - 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 {{ .annotations.asset_token | default "N/A" }}'
      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 {{ .annotations.asset_token | default "N/A" }}'
      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 {{ .annotations.asset_token | default \"N/A\"
```

```
}}",
        "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",
      "filters": [
        "entity.system.os == 'linux'",
        "entity.system.arch == 'arm'",
        "entity.system.arm_version == 7"
      ],
      "headers": {
        "Authorization": "Bearer {{ .annotations.asset_token | default \"N/A\"
}}",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    },
    {
      "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 {{ .annotations.asset_token | default \"N/A\"
}}",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    }
  ]
 }
}
```

Asset example: Single build (deprecated)

This example shows the resource definition for a dynamic runtime asset with a single build:

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 {{ .annotations.asset_token | default "N/A" }}'
    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 {{ .annotations.asset_token | default \"N/A\" }}",
        "X-Forwarded-For": "client1, proxy1, proxy2"
      }
    }
  }
```

Dynamic runtime asset build evaluation

For each build provided in a dynamic runtime asset, Sensu will evaluate any defined filters to determine whether any build matches the agent or backend service's environment. If all filters specified on a build evaluate to `true`, that build is considered a match. For dynamic runtime assets with multiple builds, only the first build that matches will be downloaded and installed.

Dynamic runtime asset build download

Sensu downloads the dynamic runtime asset build on the host system where the asset contents are needed to execute the requested command. For example, if a check definition references a dynamic runtime asset, the Sensu agent that executes the check will download the asset the first time it executes the check. The dynamic runtime asset build the agent downloads will depend on the filter rules

associated with each build defined for the asset.

Sensu backends follow a similar process when pipeline elements (filters, mutators, and handlers) request dynamic runtime asset installation as part of operation.

> **NOTE**: *Dynamic runtime asset builds are not downloaded until they are needed for command execution.*

When Sensu finds a matching build, it downloads the build artifact from the specified URL. If the asset definition includes headers, they are passed along as part of the HTTP request. If the downloaded artifact's SHA512 checksum matches the checksum provided by the build, it is unpacked into the Sensu service's local cache directory.

Set the backend or agent's local cache path with the `--cache-dir` flag. Disable dynamic runtime assets for an agent with the agent `--disable-assets` configuration flag.

> **NOTE**: *Dynamic runtime asset builds are unpacked into the cache directory that is configured with the `--cache-dir` flag.*

Use the `--assets-rate-limit` and `--assets-burst-limit` flags for the agent and backend to configure a global rate limit for fetching dynamic runtime assets.

Dynamic runtime asset build execution

The directory path of each dynamic runtime asset 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 dynamic runtime asset in the local cache and ensure that the contents match the configured checksum.

The following example demonstrates a use case with a Sensu check resource and an asset:

YML

```
---
type: Asset
api_version: core/v2
metadata:
  name: sensu-prometheus-collector
  namespace: default
```

```yaml
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
```

JSON

```json
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-email-handler",
    "namespace": "default"
  },
  "spec": {
    "builds": [
      {
```

```json
      "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"
    ]
  }
}
```

Dynamic runtime asset format specification

Sensu expects a dynamic runtime asset to be a tar archive (optionally gzipped) that contains one or more executables within a bin folder. Any scripts or executables should be within a `bin/` folder in the archive. Read the Sensu Go Plugin template for an example dynamic runtime asset and Bonsai configuration.

The following are injected into the execution context:

- `{PATH_TO_ASSET}/bin` is injected into the `PATH` environment variable
- `{PATH_TO_ASSET}/lib` is injected into the `LD_LIBRARY_PATH` environment variable
- `{PATH_TO_ASSET}/include` is injected into the `CPATH` environment variable

> **NOTE**: *You cannot create a dynamic runtime asset by creating an archive of an existing project (as in previous versions of Sensu for plugins from the Sensu Plugins community). Follow the steps outlined in Contributing Assets for Existing Ruby Sensu Plugins, a Sensu Discourse guide. For further examples of Sensu users who have added the ability to use a community plugin as a dynamic runtime asset, read this Discourse post.*

Default cache directory

system	sensu-backend	sensu-agent
Linux	`/var/cache/sensu/sensu-backend`	`/var/cache/sensu/sensu-agent`
Windows	N/A	`C:\ProgramData\sensu\cache\sensu-agent`

If the requested dynamic runtime asset is not in the local cache, it is downloaded from the asset URL. The Sensu backend acts as an index of dynamic runtime asset builds, and does not provide storage or hosting for the build artifacts. Sensu expects dynamic runtime assets to be retrieved over HTTP or HTTPS.

Example dynamic runtime asset structure
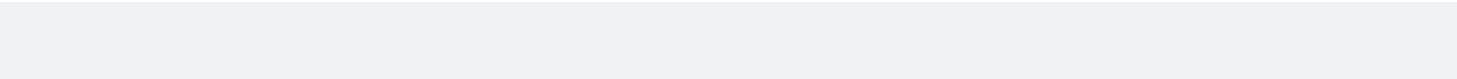
```
sensu-example-handler_1.0.0_linux_amd64
├── CHANGELOG.md
├── LICENSE
├── README.md
└── bin
   └── my-check.sh
└── lib
└── include
```

Dynamic runtime asset path

When you download and install a dynamic runtime asset, the asset files are saved to a local path on disk. Most of the time, you won't need to know this path — except in cases where you need to provide the full path to dynamic runtime asset files as part of a command argument.

The dynamic runtime asset directory path includes the asset's checksum, which changes every time underlying asset artifacts are updated. This would normally require you to manually update the commands for any of your checks, handlers, hooks, or mutators that consume the dynamic runtime asset. However, because the dynamic runtime asset directory path is exposed to asset consumers via environment variables and the `assetPath` custom function for token substitution, you can avoid these manual updates.

You can retrieve the dynamic runtime asset's path as an environment variable in the `command` context for checks, handlers, hooks, and mutators. Token substitution with the `assetPath` custom function is only available for check and hook commands.

Environment variables for dynamic runtime asset paths

For each dynamic runtime asset, a corresponding environment variable will be available in the `command` context.

Sensu generates the environment variable name by capitalizing the dynamic runtime asset name, replacing any special characters with underscores, and appending the `_PATH` suffix. The value of the variable will be the path on disk where the dynamic runtime asset build has been unpacked.

For example, for a Sensu Windows agent, the environment variable path for the dynamic runtime asset `sensu-plugins-windows` would be:

```
%SENSU_PLUGINS_WINDOWS_PATH%/include/config.yaml
```

The Windows console environment interprets the content between the paired `%` characters as an environment variable name and will substitute the value of that environment variable.

> **NOTE**: *The Sensu Windows agent uses* `cmd.exe` *for the check execution environment. For all other operating systems, the Sensu agent uses the Bourne shell (sh) and* `${VARIABLE_NAME}` *shell syntax*.

Token substitution for dynamic runtime asset paths

The `assetPath` token subsitution function allows you to substitute the dynamic runtime asset's local path on disk, so you will not need to manually update your check or hook commands every time the asset is updated.

> **NOTE**: *The* `assetPath` *function is only available where token substitution is available: the* `command` *attribute of a check or hook resource. If you want to access a dynamic runtime asset path in a handler or mutator command, you must use the environment variable*.

For example, you can reference the dynamic runtime asset `sensu-plugins-windows` from your check or hook resources using either the environment variable or the `assetPath` function:

- `%SENSU_PLUGINS_WINDOWS_PATH%/include/config.yaml`

- `${{assetPath "sensu-plugins-windows"}}/include/config.yaml`

When running PowerShell plugins on Windows, the exit status codes that Sensu captures may not match the expected values. To correctly capture exit status codes from PowerShell plugins distributed as dynamic runtime assets, use the asset path to construct the command:

YML

```yml
---
type: CheckConfig
api_version: core/v2
metadata:
  namespace: default
  name: win-cpu-check
spec:
  command: powershell.exe -ExecutionPolicy ByPass -f %{{assetPath "sensu-plugins-
windows"}}%\bin\check-windows-cpu-load.ps1 90 95
```

```yaml
  subscriptions:
  - windows
  handlers:
  - slack
  - email
  runtime_assets:
  - sensu-plugins-windows
  interval: 10
  publish: true
```

JSON

```json
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "win-cpu-check",
    "namespace": "default"
  },
  "spec": {
    "command": "powershell.exe -ExecutionPolicy ByPass -f %{{assetPath \"sensu-plugins-windows\"}}%\\bin\\check-windows-cpu-load.ps1 90 95",
    "subscriptions": [
      "windows"
    ],
    "handlers": [
      "slack",
      "email"
    ],
    "runtime_assets": [
      "sensu-plugins-windows"
    ],
    "interval": 10,
    "publish": true
  }
}
```

> **NOTE**: In this example, the check command uses the Windows console syntax for accessing the environment variables used to configure the PowerShell command line arguments.

Asset hello world Bourne shell example

In this example, you'll run a script that outputs `Hello World` :

```
hello-world.sh

#!/bin/sh

STRING="Hello World"

echo $STRING

if [ $? -eq 0 ]; then
    exit 0
else
    exit 2
fi
```

The first step is to ensure that your directory structure is in place. As noted in Example dynamic runtime asset structure, your script could live in three potential directories in the project: `/bin` , `/lib` , or `/include` . For this example, put your script in the `/bin` directory.

1. Create the directory `sensu-go-hello-world` :

    ```
    mkdir sensu-go-hello-world
    ```

2. Navigate to the `sensu-go-hello-world` directory:

    ```
    cd sensu-go-hello-world
    ```

3. Create the directory `/bin` :

```
mkdir bin
```

4. Copy the script into the `/bin` directory:

```
cp hello-world.sh bin/
```

5. Confirm that the script is in the `/bin` directory:

```
tree
```

The response should list the `hello-world.sh` script in the `/bin` directory:

```
.
└── bin
    └── hello-world.sh
```

If you receive a `command not found` response, install `tree` and run the command again.

6. Make sure that the script is marked as executable:

```
chmod +x bin/hello-world.sh
```

If you do not receive a response, the command was successful.

Now that the script is in the directory, move on to the next step: packaging the `sensu-go-hello-world` directory as a dynamic runtime asset tarball.

Package the dynamic runtime asset

Dynamic runtime assets are archives, so packaging the asset requires creating a tar.gz archive of your project.

1. Navigate to the directory you want to tar up.

2. Create the tar.gz archive:

```
tar -C sensu-go-hello-world -cvzf sensu-go-hello-world-0.0.1.tar.gz .
```

3. Generate a SHA512 sum for the tar.gz archive (this is required for the dynamic runtime asset to work):

```
sha512sum sensu-go-hello-world-0.0.1.tar.gz | tee sha512sum.txt
dbfd4a714c0c51c57f77daeb62f4a21141665ae71440951399be2d899bf44b3634dad2e6f2516f
ff1ef4b154c198b9c7cdfe1e8867788c820db7bb5bcad83827 sensu-go-hello-world-
0.0.1.tar.gz
```

From here, you can host your dynamic runtime asset wherever you'd like. To make the asset available via Bonsai, you'll need to host it on GitHub. Learn more in The "Hello World" of Sensu Assets on Discourse.

To host your dynamic runtime asset on a different platform like Gitlab or Bitbucket, upload your asset there. You can also use Artifactory or even Apache or Nginx to serve your asset. All that's required for your dynamic runtime asset to work is the URL to the asset and the SHA512 sum for the asset to be downloaded.

Asset specification

Top-level attributes

type	
description	Top-level attribute that specifies the `sensuctl create` resource type. Dynamic runtime assets should always be type `Asset` .
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .

type	String

YML

example	

```yml
type: Asset
```

JSON

```json
{
  "type": "Asset"
}
```

api_version

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

YML

example	

```yml
api_version: core/v2
```

JSON

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

metadata

description	Top-level collection of metadata about the dynamic runtime asset,

including `name` , `namespace` , and `created_by` as well as custom `labels` and `annotations` . The `metadata` map is always at the top level of the asset definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. Read metadata attributes for details.

required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create` .
type	Map of key-value pairs

example

YML

```yml
metadata:
  name: check_script
  namespace: default
  created_by: admin
  labels:
    region: us-west-1
  annotations:
    playbook: www.example.url
```

JSON

```json
{
  "metadata": {
    "name": "check_script",
    "namespace": "default",
    "created_by": "admin",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "playbook": "www.example.url"
    }
  }
}
```

spec

description	Top-level map that includes the dynamic runtime asset spec attributes.
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example (multiple builds)	YML

```yaml
spec:
  builds:
  - url: http://example.com/asset-linux-amd64.tar.gz
    sha512:
487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a
72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fcc
b574c1a5d3
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    headers:
      Authorization: Bearer {{ .annotations.asset_token |
default "N/A" }}
      X-Forwarded-For: client1, proxy1, proxy2
  - url: http://example.com/asset-linux-armv7.tar.gz
    sha512:
70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d1538374066
a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f7
5670a8c68b
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'arm'
    - entity.system.arm_version == 7
    headers:
      Authorization: Bearer {{ .annotations.asset_token |
default "N/A" }}
      X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```json
{
  "spec": {
    "builds": [
      {
```

```
        "url": "http://example.com/asset-linux-
amd64.tar.gz",
        "sha512":
"487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58
a72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fc
cb574c1a5d3",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'"
        ],
        "headers": {
          "Authorization": "Bearer {{
.annotations.asset_token | default \"N/A\" }}",
          "X-Forwarded-For": "client1, proxy1, proxy2"
        }
      },
      {
        "url": "http://example.com/asset-linux-
armv7.tar.gz",
        "sha512":
"70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d153837406
6a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f
75670a8c68b",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'arm'",
          "entity.system.arm_version == 7"
        ],
        "headers": {
          "Authorization": "Bearer {{
.annotations.asset_token | default \"N/A\" }}",
          "X-Forwarded-For": "client1, proxy1, proxy2"
        }
      }
    ]
  }
}
```

	YML
example (single	

build, deprecated)

```yaml
spec:
  url: http://example.com/asset.tar.gz
  sha512:
4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3
ef3c2e8d154812246e5dda4a87450576b2c58ad9ab40c9e2edc31b288d0
66b195b21b
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
  headers:
    Authorization: Bearer {{ .annotations.asset_token |
default "N/A" }}
    X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```json
{
  "spec": {
    "url": "http://example.com/asset.tar.gz",
    "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a
3ef3c2e8d154812246e5dda4a87450576b2c58ad9ab40c9e2edc31b288d
066b195b21b",
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ],
    "headers": {
      "Authorization": "Bearer {{ .annotations.asset_token
| default \"N/A\" }}",
      "X-Forwarded-For": "client1, proxy1, proxy2"
    }
  }
}
```

Metadata attributes

name

description	Unique name of the dynamic runtime asset, validated with Go regex `\A[\w\.\-]+\z` .
required	true
type	String
example	YML

```yml
name: check_script
```

JSON

```json
{
  "name": "check_script"
}
```

namespace

description	Sensu RBAC namespace that the dynamic runtime asset belongs to.
required	false
type	String
default	`default`
example	YML

```yml
namespace: production
```

JSON

```json
{
  "namespace": "production"
}
```

created_by

description	Username of the Sensu user who created the dynamic runtime asset or last updated the asset. Sensu automatically populates the `created_by` field when the dynamic runtime asset is created or updated.
required	false
type	String
| example | **YML**<br><br>```yml
created_by: admin
```<br><br>**JSON**<br><br>```json
{
  "created_by": "admin"
}
``` |

labels

description	Custom attributes to include with observation data in events that you can use for response and web UI view filtering. If you include labels in your event data, you can filter API responses, sensuctl responses, and web UI views based on them. In other words, labels allow you to create meaningful groupings for your data. Limit labels to metadata you need to use for response filtering. For complex, non-identifying metadata that you will *not* need to use in response filtering, use annotations rather than labels.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores and must start with a letter. Values can be any valid UTF-8 string.

default	`null`

YML

```
labels:
  environment: development
  region: us-west-2
```

example	

JSON

```
{
  "labels": {
    "environment": "development",
    "region": "us-west-2"
  }
}
```

annotations

description	Non-identifying metadata to include with observation data in events that you can access with event filters. You can use annotations to add data that's meaningful to people or external tools that interact with Sensu. In contrast to labels, you cannot use annotations in API response filtering, sensuctl response filtering, or web UI views.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

YML

example	

```
annotations:
  managed-by: ops
  playbook: www.example.url
```

JSON

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

Spec attributes

builds	
description	List of dynamic runtime asset builds used to define multiple artifacts that provide the named asset.
required	true, if `url`, `sha512` and `filters` are not provided
type	Array
example	**YML**

```yaml
builds:
- url: http://example.com/asset-linux-amd64.tar.gz
  sha512:
487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58a
72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fcc
b574c1a5d3
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
- url: http://example.com/asset-linux-armv7.tar.gz
  sha512:
70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d1538374066
a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f7
5670a8c68b
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'arm'
  - entity.system.arm_version == 7
```

JSON

```json
{
  "builds": [
    {
      "url": "http://example.com/asset-linux-amd64.tar.gz",
      "sha512":
"487ab34b37da8ce76d2657b62d37b35fbbb240c3546dd463fa0c37dc58
a72b786ef0ca396a0a12c8d006ac7fa21923e0e9ae63419a4d56aec41fc
cb574c1a5d3",
      "filters": [
        "entity.system.os == 'linux'",
        "entity.system.arch == 'amd64'"
      ]
    },
    {
      "url": "http://example.com/asset-linux-armv7.tar.gz",
      "sha512":
"70df8b7e9aa36cf942b972e1781af04815fa560441fcdea1d153837406
6a4603fc5566737bfd6c7ffa18314edb858a9f93330a57d430deeb7fd6f
75670a8c68b",
      "filters": [
        "entity.system.os == 'linux'",
        "entity.system.arch == 'arm'",
        "entity.system.arm_version == 7"
      ]
    }
  ]
}
```

url	
description	URL location of the dynamic runtime asset. You can use token substitution in the URLs of your asset definitions so each backend or agent can download dynamic runtime assets from the appropriate URL without duplicating your assets (for example, if you want to host your assets at different datacenters).

required	true, unless `builds` are provided
type	String

YML

example	

```yml
url: http://example.com/asset.tar.gz
```

JSON

```json
{
  "url": "http://example.com/asset.tar.gz"
}
```

sha512

description	Checksum of the dynamic runtime asset.
required	true, unless `builds` are provided
type	String

YML

example	

```yml
sha512: 4f926bf4328...
```

JSON

```json
{
  "sha512": "4f926bf4328..."
}
```

filters

description	Set of Sensu query expressions used to determine if the dynamic runtime asset should be installed. If multiple expressions are included,

each expression must return `true` for Sensu to install the asset.

Filters for *check* dynamic runtime assets should match agent entity platforms. Filters for *handler* and *filter* dynamic runtime assets should match your Sensu backend platform. You can create asset filter expressions using any supported <u>entity.system attributes</u>, including `os`, `arch`, `platform`, and `platform_family`.

> **PRO TIP**: *Dynamic runtime asset filters let you reuse checks across platforms safely. Assign dynamic runtime assets for multiple platforms to a single check, and rely on asset filters to ensure that only the appropriate asset is installed on each agent.*

required	false
type	Array

example	**YML**

```yml
filters:
- entity.system.os=='linux'
- entity.system.arch=='amd64'
```

JSON

```json
{
  "filters": [
    "entity.system.os=='linux'",
    "entity.system.arch=='amd64'"
  ]
}
```

headers

description	HTTP headers to apply to dynamic runtime asset retrieval requests. You can use headers to access secured dynamic runtime assets. For headers that require multiple values, separate the values with a comma. You can use <u>token substitution</u> in your dynamic runtime asset headers

(for example, to include secure information for authentication).

required	false
type	Map of key-value string pairs
example	

YML

```yml
headers:
  Authorization: Bearer {{ .annotations.asset_token | default "N/A" }}
  X-Forwarded-For: client1, proxy1, proxy2
```

JSON

```json
{
  "headers": {
    "Authorization": "Bearer {{ .annotations.asset_token | default \"N/A\" }}",
    "X-Forwarded-For": "client1, proxy1, proxy2"
  }
}
```

Dynamic runtime asset filters based on entity.system attributes

Use the entity.system attributes in dynamic runtime asset filters to specify which systems and configurations an asset or asset builds can be used with.

For example, the Sensu Go Ruby Runtime dynamic runtime asset definition includes several builds, each with filters for several `entity.system` attributes:

YML

```yml
---
type: Asset
api_version: core/v2
metadata:
```

```yaml
  name: sensu-ruby-runtime
  labels:
  annotations:
    io.sensu.bonsai.url: https://bonsai.sensu.io/assets/sensu/sensu-ruby-runtime
    io.sensu.bonsai.api_url: https://bonsai.sensu.io/api/v1/assets/sensu/sensu-ruby-runtime
    io.sensu.bonsai.tier: Community
    io.sensu.bonsai.version: 0.0.10
    io.sensu.bonsai.namespace: sensu
    io.sensu.bonsai.name: sensu-ruby-runtime
    io.sensu.bonsai.tags: ''
spec:
  builds:
  - url:
https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-runtime_0.0.10_ruby-2.4.4_centos6_linux_amd64.tar.gz
    sha512:
cbee19124b7007342ce37ff9dfd4a1dde03beb1e87e61ca2aef606a7ad3c9bd0bba4e53873c07afa5ac46b0861967a9224511b4504dadb1a5e8fb687e9495304
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    - entity.system.platform_family == 'rhel'
    - parseInt(entity.system.platform_version.split('.')[0]) == 6
  - url:
https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-runtime_0.0.10_ruby-2.4.4_debian_linux_amd64.tar.gz
    sha512:
a28952fd93fc63db1f8988c7bc40b0ad815eb9f35ef7317d6caf5d77ecfbfd824a9db54184400aa0c81c29b34cb48c7e8c6e3f17891aaf84cafa3c134266a61a
    filters:
    - entity.system.os == 'linux'
    - entity.system.arch == 'amd64'
    - entity.system.platform_family == 'debian'
  - url:
https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-runtime_0.0.10_ruby-2.4.4_alpine_linux_amd64.tar.gz
    sha512:
8d768d1fba545898a8d09dca603457eb0018ec6829bc5f609a1ea51a2be0c4b2d13e1aa46139ecbb04873449e4c76f463f0bdfbaf2107caf37ab1c8db87d5250
    filters:
    - entity.system.os == 'linux'
```

```
      - entity.system.arch == 'amd64'
      - entity.system.platform == 'alpine'
      - entity.system.platform_version.split('.')[0] == '3'
```

JSON

```json
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-ruby-runtime",
    "labels": null,
    "annotations": {
      "io.sensu.bonsai.url": "https://bonsai.sensu.io/assets/sensu/sensu-ruby-
runtime",
      "io.sensu.bonsai.api_url": "https://bonsai.sensu.io/api/v1/assets/sensu/sensu-
ruby-runtime",
      "io.sensu.bonsai.tier": "Community",
      "io.sensu.bonsai.version": "0.0.10",
      "io.sensu.bonsai.namespace": "sensu",
      "io.sensu.bonsai.name": "sensu-ruby-runtime",
      "io.sensu.bonsai.tags": ""
    }
  },
  "spec": {
    "builds": [
      {
        "url":
"https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
runtime_0.0.10_ruby-2.4.4_centos6_linux_amd64.tar.gz",
        "sha512":
"cbee19124b7007342ce37ff9dfd4a1dde03beb1e87e61ca2aef606a7ad3c9bd0bba4e53873c07afa5ac
46b0861967a9224511b4504dadb1a5e8fb687e9495304",
        "filters": [
          "entity.system.os == 'linux'",
          "entity.system.arch == 'amd64'",
          "entity.system.platform_family == 'rhel'",
          "parseInt(entity.system.platform_version.split('.')[0]) == 6"
        ]
      },
      {
        "url":
```

```
      "https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
      runtime_0.0.10_ruby-2.4.4_debian_linux_amd64.tar.gz",
            "sha512":
      "a28952fd93fc63db1f8988c7bc40b0ad815eb9f35ef7317d6caf5d77ecfbfd824a9db54184400aa0c81
      c29b34cb48c7e8c6e3f17891aaf84cafa3c134266a61a",
            "filters": [
              "entity.system.os == 'linux'",
              "entity.system.arch == 'amd64'",
              "entity.system.platform_family == 'debian'"
            ]
          },
          {
            "url":
      "https://assets.bonsai.sensu.io/5123017d3dadf2067fa90fc28275b92e9b586885/sensu-ruby-
      runtime_0.0.10_ruby-2.4.4_alpine_linux_amd64.tar.gz",
            "sha512":
      "8d768d1fba545898a8d09dca603457eb0018ec6829bc5f609a1ea51a2be0c4b2d13e1aa46139ecbb048
      73449e4c76f463f0bdfbaf2107caf37ab1c8db87d5250",
            "filters": [
              "entity.system.os == 'linux'",
              "entity.system.arch == 'amd64'",
              "entity.system.platform == 'alpine'",
              "entity.system.platform_version.split('.')[0] == '3'"
            ]
          }
        ]
      }
    }
```

In this example, if you install the dynamic runtime asset on a system running Linux AMD64 Alpine
version 3.xx, Sensu will ignore the first two builds and install the third.

> **NOTE**: Sensu downloads and installs the first build whose filter expressions all evaluate as `true`.
> If your system happens to match all of the filters for more than one build of a dynamic runtime
> asset, Sensu will only install the first build.

All of the dynamic runtime asset filter expressions must evaluate as true for Sensu to download and
install the asset and run the check, handler, or filter that references the asset.

To continue this example, if you try to install the dynamic runtime asset on a system running Linux AMD64 Alpine version 2.xx, the `entity.system.platform_version` argument will evaluate as `false`. In this case, the asset will not be downloaded and the check, handler, or filter that references the asset will fail to run.

Add dynamic runtime asset filters to specify that an asset is compiled for any of the entity.system attributes, including operating system, platform, platform version, and architecture. Then, you can rely on dynamic runtime asset filters to ensure that you install only the appropriate asset for each of your agents.

Share an asset on Bonsai

Share your open-source dynamic runtime assets on Bonsai and connect with the Sensu community. Bonsai supports dynamic runtime assets hosted on GitHub and released using GitHub releases. For more information about creating Sensu plugins, read the plugins reference.

Bonsai requires a `bonsai.yml` configuration file in the root directory of your repository that includes the project description, platforms, asset filenames, and SHA-512 checksums. For a Bonsai-compatible dynamic runtime asset template using Go and GoReleaser, review the Sensu Go plugin skeleton.

To share your dynamic runtime asset on Bonsai, log in to Bonsai with your GitHub account and authorize Sensu. After you are logged in, you can register your dynamic runtime asset on Bonsai by adding the GitHub repository, a description, and tags. Make sure to provide a helpful README for your dynamic runtime asset with configuration examples.

`bonsai.yml` example

```
---
description: "#{repo}"
builds:
- platform: "linux"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -   "entity.system.os == 'linux'"
  -   "entity.system.arch == 'amd64'"

- platform: "Windows"
```

```
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_windows_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'windows'"
  -  "entity.system.arch == 'amd64'"
```

`bonsai.yml` specification

description	
description	Project description.
required	true
type	String
example	```\ndescription: "#{repo}"\n```

builds	
description	Array of dynamic runtime asset details per platform.
required	true
type	Array
example	```\nbuilds:\n- platform: "linux"\n arch: "amd64"\n asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"\n sha_filename: "#{repo}_#{version}_sha512-checksums.txt"\n filter:\n - "entity.system.os == 'linux'"\n - "entity.system.arch == 'amd64'"\n```

Builds specification

platform

description	Platform supported by the dynamic runtime asset.
required	true
type	String
| example | ```yaml
- platform: "linux"
``` |

arch

description	Architecture supported by the dynamic runtime asset.
required	true
type	String
| example | ```yaml
arch: "amd64"
``` |

asset_filename

description	File name of the archive that contains the dynamic runtime asset.
required	true
type	String
| example | ```yaml
asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
``` |

sha_filename	
description	SHA-512 checksum for the dynamic runtime asset archive.
required	true
type	String
| example | ```
sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
``` |

filter	
description	Filter expressions that describe the operating system and architecture supported by the asset.
required	false
type	Array
| example | ```
filter:
-  "entity.system.os == 'linux'"
-  "entity.system.arch == 'amd64'"
``` |

Delete dynamic runtime assets

Delete dynamic runtime assets with the `/assets (DELETE)` endpoint or via `sensuctl` (`sensuctl asset delete`).When you remove a dynamic runtime 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 dynamic runtime asset.Failure to do so will result in errors like `sh: asset.sh: command not found` .

Errors as a result of failing to remove the dynamic runtime asset from checks and hooks will surface in the event data.Errors as a result of failing to remove the dynamic runtime asset reference on a mutator, handler, or filter will only surface in the backend logs.

Deleting a dynamic runtime asset does not delete the archive or downloaded files on disk. You must remove the archive and downloaded files from the asset cache manually.

Plugins reference

Sensu plugins provide executable scripts or other programs that you can use as Sensu checks, handlers, and mutators. Sensu plugins must comply with the following specification:

- Accept input/data via `STDIN` (handler and mutator plugins only)
  - Optionally able to parse a JSON data payload (that is, observation data in an event)
- Output data to `STDOUT` or `STDERR`
- Produce an exit status code to indicate state:
  - `0` indicates `OK`
  - `1` indicates `WARNING`
  - `2` indicates `CRITICAL`
  - exit status codes other than `0`, `1`, or `2` indicate an unknown or custom status
- Optionally able to parse command line arguments to modify plugin behavior

Supported programming languages

You can use any programming language that can satisfy the Sensu plugin specification requirements — which is nearly any programming language in the world — to write Sensu plugins.

Using plugins written in programming languages other than Go requires you to install the corresponding runtime. For example, to use a Ruby plugin with Sensu Go, you must install the Sensu Go Ruby Runtime asset.

Plugin execution

All plugins are executed by the Sensu backend. Plugins must be executable files that are discoverable on the Sensu system (that is, installed in a system `$PATH` directory) or referenced with an absolute path (for example, `/opt/path/to/my/plugin`).

> **NOTE**: By default, Sensu installer packages will modify the system `$PATH` for the Sensu

processes to include `/etc/sensu/plugins`. As a result, executable scripts (for example, plugins) located in `/etc/sensu/plugins` will be valid commands. This allows command attributes to use relative paths for Sensu plugin commands, such as `"command": "http-check --url https://sensu.io"`.

Go plugin example

The following example shows the structure for a very basic Sensu Go plugin.

```go
package main

import (
    "fmt"
    "log"

    "github.com/sensu-community/sensu-plugin-sdk/sensu"
    "github.com/sensu/sensu-go/types"
)

// Config represents the check plugin config.
type Config struct {
    sensu.PluginConfig
    Example string
}

var (
    plugin = Config{
        PluginConfig: sensu.PluginConfig{
            Name:     "check_name",
            Short:    "Description for check_name",
            Keyspace: "sensu.io/plugins/check_name/config",
        },
    }

    options = []*sensu.PluginConfigOption{
        &sensu.PluginConfigOption{
            Path:     "example",
            Env:      "CHECK_EXAMPLE",
            Argument: "example",
```

```
        Shorthand: "e",
        Default:   "",
        Usage:     "An example string configuration option",
        Value:     &plugin.Example,
    },
      }
)


func main() {
    check := sensu.NewGoCheck(&plugin.PluginConfig, options, checkArgs, executeCheck,
false)
    check.Execute()
}


func checkArgs(event *types.Event) (int, error) {
    if len(plugin.Example) == 0 {
  return sensu.CheckStateWarning, fmt.Errorf("--example or CHECK_EXAMPLE
environment variable is required")
    }
    return sensu.CheckStateOK, nil
}


func executeCheck(event *types.Event) (int, error) {
    log.Println("executing check with --example", plugin.Example)
    return sensu.CheckStateOK, nil
}
```

To create this scaffolding for a Sensu Go check, handler, mutator, or sensuctl plugin, use the Sensu Plugin Tool along with a default plugin template.The plugin template repositories wrap the Sensu Plugin SDK, which provides the framework for building Sensu Go plugins.

For a step-by-step walkthrough, read How to publish an asset with the Sensu Go SDK — you'll learn how to create a check plugin and a handler plugin with the Sensu Plugin SDK.You can also watch our 30-minute webinar, Intro to assets with the Sensu Go SDK, and learn to build a check plugin for Sensu Go.

Ruby plugin example

The following example demonstrates a very basic Sensu plugin in the Ruby programming language.

```ruby
#!/usr/bin/env ruby
#
require 'json'

# Read the incoming JSON data from STDIN
event = JSON.parse(STDIN.read, :symbolize_names => true)

# Create an output object using Ruby string interpolation
output = "The check named #{event[:check][:name]} generated the following output: #
{event[:output]}"

# Convert the mutated event data back to JSON and output it to STDOUT
puts output
```

NOTE: This example is intended as a starting point for building a basic custom plugin in Ruby. It does not provide functionality.

Supported Integrations

Sensu integrations include plugins, libraries, and runtimes that extend Sensu's functionality and allow you to automate your monitoring and observability workflows. You can also rely on Sensu's integrations to get work done with Sensu as part of your existing workflows.

Integrations are service-specific and have different setup and configuration requirements. Each integration has self-contained documentation with in-depth information about how to install and use it. Many of the supported integrations include curated quick-start templates that you only need to edit to match your configuration.

Although this category focuses on our most popular supported integrations, you can find more supported-, Enterprise-, and community-tier integrations at Bonsai, the Sensu asset hub.

Alerting and incident management

- ¬ Email
- ¬ Jira
- ¬ PagerDuty
- ¬ ServiceNow
- ¬ Slack

Auto-remediation

- ¬ Ansible
- ¬ Rundeck
- ¬ SaltStack

Deregistration

- ¬ Chef
- ¬

- ¬ EC2
- ¬ Puppet

Time-series and long-term event storage

- ¬ Elasticsearch
- ¬ Graphite
- ¬ InfluxDB
- ¬ OpenTSDB
- ¬ Prometheus
- ¬ Sumo Logic
- ¬ TimescaleDB
- ¬ Wavefront

Ansible integration

The Sensu Ansible Handler plugin is a Sensu handler that launches Ansible Tower job templates for automated remediation based on Sensu event data.

NOTE: The Sensu Ansible Handler plugin is an example of Sensu's auto-remediation integrations. To find more integrations, search Bonsai, the Sensu asset hub.

Features

The Sensu Ansible Handler plugin supports both Ansible Tower and Ansible AWX implementations of the Ansible Tower API, authenticating using Ansible Tower API tokens.

- Specify a default Ansible Tower job template for remediation actions for all checks and use check annotations to override the default as needed on a check-by check-basis.

- Automatically limit Ansible jobs to the host that matches the Sensu entity name encoded in a Sensu event.

- Optional job template requests: use Sensu check annotations to specify a set of Ansible Tower job template requests to run for matching Sensu event occurrence and severity conditions.

- Keep your Ansible Tower host and auth token secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Ansible Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing Ansible Tower workflows.

You can also add the Sensu Ansible Handler plugin with a dynamic runtime asset from Bonsai, the

Sensu asset hub, to build your own workflow or integrate Sensu with your existing Ansible workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Configuration management

Use the official Sensu Go Ansible Collection for configuration management for your Sensu instance. The documentation site includes installation instructions, example playbooks, and module references

Chef integration

The Sensu Chef Handler plugin is a Sensu handler that deletes a Sensu entity with a failing keepalive check when the entity's corresponding Chef node no longer exists.

> **NOTE**: *The Sensu Chef Handler plugin is an example of Sensu's deregistration integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Use Sensu annotations to override Sensu entity names with corresponding Chef node names.

- Keep your sensitive API authentication information secure with Sensu environment variables and secrets management.

Get the plugin

Add the Sensu Chef Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Chef workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Configuration management

Use the official Chef Cookbook for Sensu Go for configuration management for your Sensu instance.

EC2 integration

The Sensu EC2 Handler plugin is a Sensu handler that checks an AWS EC2 instance and removes it from Sensu if it is not in one of the specified states.

> **NOTE**: *The Sensu EC2 Handler plugin is an example of Sensu's deregistration integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Tunable arguments: use Sensu annotations to set custom instance ID, instance ID labels, timeouts, and more in EC2.

- Specify custom values for Sensu event metric points via metric tags.

- Keep your AWS EC2 API token, username, and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu EC2 Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing AWS EC2 workflows.

You can also add the Sensu EC2 Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing EC2 workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

- Set up a limited service account with the access and permissions required to automatically remove AWS EC2 instances using the Sensu EC2 Handler integration.

Elasticsearch integration

The [Sensu Elasticsearch Handler plugin](#) is a Sensu [handler](#) that sends observation data from Sensu events and metrics to Elasticsearch. With this handler, the Sensu observation data you send to Elasticsearch is available for indexing and visualization in Kibana.

NOTE: *The Sensu Elasticsearch Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search [Bonsai, the Sensu asset hub](#).*

Features

- Query metrics points within Elasticsearch: the handler automatically mutates metrics data by creating a top-level object with metric point names and their associated values.

- Index entire events for searching within Kibana.

- Use daily, weekly, monthly, and yearly index specification (for example, sensu_events-2020-11-10).

- Omit the transmission of certain redundant event fields to reduce the number of items indexed.

- Specify custom values for Sensu event metric points via [metric tags](#).

- Use [event-based templating](#) to include observation data from event attributes to add meaningful, actionable context.

- Keep your Elasticsearch username and password secure with Sensu [environment variables and secrets management](#).

Get the plugin

For a turnkey experience with the Sensu Elasticsearch Handler plugin, use our curated, configurable [quick-start template](#) for events and metrics data storage.

Add the Sensu Elasticsearch Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Elasticsearch workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Email integration

The Sensu Email Handler plugin is a Sensu handler that sends email alerts based on your event data. With this handler, Sensu can send email messages to the email addresses you specify based on event data generated by your Sensu checks.

> **NOTE**: The Sensu Email Handler plugin is an example of Sensu's alerting integrations. To find more integrations, search Bonsai, the Sensu asset hub.

Features

The Sensu Email Handler plugin supports the login authentication mechanisms required for use with Google Mail, Office 365, and other standards-based email providers and transports.

- Use event-based templating to include observation data from event attributes to add meaningful, actionable context to your email alert messages.
- Keep your email provider username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Email Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and send email alerts.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Email Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing email workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

Walk through adding and configuring the Sensu Email Handler asset in the Send email alerts with the

[Sensu Go Email Handler](#) guide.

Graphite integration

The Sensu Graphite Handler plugin is a Sensu handler that sends Sensu metrics to the time-series database Graphite so you can store, instrument, and visualize Sensu metrics data.

> **NOTE**: *The Sensu Graphite Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Transform metrics to Graphite format: extract and transform the metrics you collect from different sources in formats like Influx, Nagios, and OpenTSDB and populate them into Graphite.

- Specify custom values for Sensu event metric points via metric tags.

- Keep your Graphite host and port secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Graphite Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in Graphite.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Graphite Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

InfluxDB integration

The Sensu InfluxDB Handler plugin is a Sensu handler that sends Sensu metrics to the time-series database InfluxDB so you can store, instrument, and visualize Sensu metrics data. You can also use the Sensu InfluxDB Handler integration to create metrics from Sensu status check results for long-term storage in InfluxDB.

> **NOTE**: *The Sensu InfluxDB Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Transform metrics to InfluxDB format: extract and transform the metrics you collect from different sources in formats like Graphite, OpenTSDB, Nagios, and Influx and populate them into InfluxDB.

- Mutate check status into metrics to be stored in InfluxDB.

- Specify custom values for Sensu event metric points via metric tags.

- Keep your InfluxDB username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu InfluxDB Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in InfluxDB.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu InfluxDB Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Jira integration

The Sensu Jira Handler plugin is a Sensu handler that creates and updates Jira issues based on observation data from Sensu events.

NOTE: The Sensu Jira Handler plugin is an example of Sensu's alerting and incident management integrations. To find more integrations, search _Bonsai, the Sensu asset hub_.

Features

- Tunable arguments: use Sensu annotations to set custom project names, issue types, resolution states, and more in Jira

- Use event-based templating to include observation data from event attributes to add meaningful, actionable context.

- Keep your Jira username, password, and API token secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Jira Handler plugin, use our curated, configurable quick-start template to send alerts based on Sensu events to Jira Service Desk.

Add the Sensu Jira Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Jira workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

OpenTSDB integration

The Sensu OpenTSDB Handler plugin is a Sensu handler that sends Sensu metrics to an OpenTSDB server via its Telnet-style API. This allows you to extract, tag, and store Sensu metrics data in an OpenTSDB database.

> **NOTE**: The Sensu OpenTSDB Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.

Features

- Transform metrics to OpenTSDB format: extract and transform the metrics you collect from different sources in formats like Graphite, Influx, and Nagios and populate them into OpenTSDB.

- Specify custom values for Sensu event metric points via metric tags.

Get the plugin

To build your own workflow or integrate Sensu with existing workflows, add the Sensu OpenTSDB Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

PagerDuty integration

The Sensu PagerDuty Handler plugin is a Sensu handler that manages PagerDuty incidents and operator alerts. With this handler, Sensu can trigger and resolve PagerDuty incidents according to the PagerDuty schedules, notifications, and escalation, response, and orchestration workflows you already have configured.

> **NOTE**: *The Sensu PagerDuty Handler plugin is an example of Sensu's alerting and incident management integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Optional severity mapping: match Sensu check statuses with PagerDuty incident severities via a JSON document.

- Use event-based templating to create deduplication key arguments to group repeated alerts into one incident and summary template arguments to make sure your PagerDuty notifications include the event data your operators need to take action.

- Authenticate and route alerts based on PagerDuty teams using check and agent annotations.

- Keep your PagerDuty integration key secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu PagerDuty Handler plugin, use our curated, configurable quick-start template for incident management to integrate Sensu with your existing PagerDuty workflows.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu PagerDuty Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

- ¬ Follow the Use dynamic runtime assets to install plugins guide to learn how to add and configure Sensu PagerDuty Handler asset.

- ¬ Demo the Sensu PagerDuty Handler integration with the Send Sensu Go alerts to PagerDuty guide.

Prometheus integrations

Sensu has two Prometheus plugins: the Prometheus Collector and the Prometheus Pushgateway Handler. Both help you get Sensu observability data into Prometheus.

> **NOTE**: The Sensu Prometheus plugins are examples of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.

Sensu Prometheus Collector

The Sensu Prometheus Collector plugin is a Sensu check that collects metrics from a Prometheus exporter or the Prometheus query API and outputs the metrics to STDOUT in Influx, Graphite, or JSON format.

Features

- Turn Sensu into a super-powered Prometheus metric poller with Sensu's publish/subscribe model and client auto-registration (discovery) capabilities.

- Configure your Sensu instance to deliver the collected metrics to a time-series database like InfluxDB or Graphite.

- Specify custom values for Sensu event metric points via metric tags.

- Keep metrics endpoint authentication information secure with Sensu environment variables and secrets management.

Sensu Prometheus Pushgateway Handler

The Sensu Prometheus Pushgateway Handler plugin is a Sensu handler that sends Sensu metrics to a Prometheus Pushgateway, which Prometheus can then scrape.

Features

-

- Collect metrics by several means, including 20-year-old Nagios plugins with perfdata, and store them in Prometheus.

- Use default Prometheus metric type, job name, and instance name or specify custom values for Sensu event metric points via metric tags.

Get the plugins

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Prometheus plugins with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

- Sensu Prometheus Collector plugin

- Sensu Prometheus Pushgateway Handler

Puppet integration

The Sensu Puppet Keepalive Handler plugin is a Sensu handler that deletes a Sensu entity with a failing keepalive check when the entity's corresponding Puppet node no longer exists or is deregistered.

> **NOTE**: *The Sensu Puppet Keepalive Handler plugin is an example of Sensu's deregistration integrations.To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Use Sensu annotations to override Sensu entity names with corresponding Puppet node names.

- Keep sensitive API authentication information secure with Sensu environment variables and secrets management.

Get the plugin

Add the Sensu Puppet Keepalive Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Puppet workflows.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Configuration management

Use the partner-supported Sensu Puppet module for configuration management for your Sensu instance.

Rundeck integration

The Sensu Rundeck Handler plugin is a Sensu handler that initiates Rundeck jobs for automated remediation based on Sensu event data.

NOTE: The Sensu Rundeck Handler plugin is an example of Sensu's auto-remediation integrations. To find more integrations, search Bonsai, the Sensu asset hub.

Features

The Sensu Rundeck Handler plugin supports both Rundeck Enterprise and Rundeck Open Source and standard job invocation or webhook invocation.

- Specify Rundeck jobs and webhooks along with trigger parameters for remediation actions for a check with Sensu annotations.
- Use event-based templating to make use of event data to specify the node to target for rememdiation.
- Keep your Rundeck auth token and webhook secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Rundeck Handler plugin, use one of our curated, configurable quick-start templates:

- Rundeck job
- Rundeck webhook

You can also add the Sensu Rundeck Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Rundeck workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

SaltStack integration

The [Sensu SaltStack Handler plugin](#) is a Sensu [handler](#) that launches SaltStack functions for automated remediation based on Sensu event data.

NOTE: *The Sensu SaltStack Handler plugin is an example of Sensu's auto-remediation integrations.To find more integrations, search [Bonsai, the Sensu asset hub](#).*

Features

The [Sensu SaltStack Handler plugin](#) supports both SaltStack Enterprise and SaltStack Open Source as well as SaltStack functions such as `service`, `state`, `saltutil`, and `grains` (including `arg` and `kwarg` arguments).

- ¬ Specify SaltStack functions and trigger parameters for remediation actions for a check with Sensu annotations.
- ¬ Use [event-based templating](#) to specify the minion to target for rememdiation based on event data.
- ¬ Keep your SaltStack username and password secure with Sensu [environment variables and secrets management](#).

Get the plugin

For a turnkey experience with the Sensu SaltStack Handler plugin, use our curated, configurable [quick-start template](#) to integrate Sensu with your existing SaltStack workflows.

You can also add the [Sensu SaltStack Handler plugin](#) with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing SaltStack workflows. [Dynamic runtime assets](#) are shareable, reusable packages that make it easier to deploy Sensu

plugins.

ServiceNow integration

The Sensu ServiceNow Handler plugin is a Sensu handler that creates and updates ServiceNow incidents and events based on observation data from Sensu events.

> **NOTE**: The Sensu ServiceNow Handler plugin is an example of Sensu's alerting and incident management integrations. To find more integrations, search _Bonsai, the Sensu asset hub_.

Features

- Automatically create a ServiceNow Configuration Item if none currently exists for a particular Sensu entity.

- Tunable arguments: use Sensu annotations to set custom incident notes, event information, Configuration Item descriptions, and more in ServiceNow.

- Use event-based templating to include observation data from event attributes to add meaningful, actionable context to ServiceNow incidents, events, and Configuration Items.

- Keep your ServiceNow username and password secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu ServiceNow Handler plugin, use one of our curated, configurable quick-start templates:

- ServiceNow Incident Management: send Sensu observability alerts to ServiceNow Incident Management.

- ServiceNow Event Management: send Sensu observability data to ServiceNow Event Management.

- ¬ ServiceNow Configuration Management Database (CMDB): register Sensu entities as configuration items in ServiceNow CMDB.

You can also add the Sensu ServiceNow Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing ServiceNow workflows.Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Slack integration

The Sensu Slack Handler plugin is a Sensu handler that sends alerts based on your event data. With this handler, Sensu can trigger alerts to the Slack channels you specify based on event data generated by your Sensu checks.

> **NOTE**: The Sensu Slack Handler plugin is an example of Sensu's alerting and incident management integrations. To find more integrations, search Bonsai, the Sensu asset hub.

Features

- Use event-based templating to include observation data from event attributes in your alerts to add meaningful, actionable context.

- Keep your Slack webhook secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Slack Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and send alerts to Slack channels.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Slack Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Slack workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

More resources

Read Send Slack alerts with handlers to learn how to add and configure the Sensu Slack Handler plugin.

Sumo Logic integration

The Sensu Sumo Logic Handler plugin is a Sensu handler that sends Sensu observability events and metrics to a Sumo Logic HTTP Logs and Metrics Source. This handler sends Sensu events as log entries, a set of metrics, or both, depending on the mode of operation you specify.

> **NOTE**: The Sensu Sumo Logic Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search *Bonsai, the Sensu asset hub*.

Features

- Query events and metrics points within Sumo Logic: the handler automatically mutates metrics data by creating a top-level object with metric point names and their associated values.

- Tunable arguments: use Sensu annotations to set Sumo Logic source name, host, and category; metric dimensions; log fields; and more.

- Use event-based templating to include observation data from event attributes to add meaningful, actionable context.

- Keep your Sumo Logic HTTP Logs and Metrics Source URL secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Sumo Logic Handler plugin, use our curated, configurable quick-start templates for event storage and metric storage to integrate Sensu with your existing workflows and send observation data to an HTTP Logs and Metrics Source.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Sumo Logic Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub, to build your own workflow or integrate Sensu with your existing Sumo Logic workflows. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

TimescaleDB integration

The Sensu TimescaleDB Handler plugin is a Sensu handler that sends Sensu metrics to the time-series database TimescaleDB so you can store, instrument, and visualize Sensu metrics data.

> **NOTE**: The Sensu TimescaleDB Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search *Bonsai, the Sensu asset hub*.

Features

- Transform metrics to TimescaleDB format: extract and transform the metrics you collect from different sources in formats like Graphite, OpenTSDB, Nagios, and Influx and populate them into TimescaleDB.

- Specify custom values for Sensu event metric points via metric tags.

Get the plugin

To build your own workflow or integrate Sensu with existing workflows, add the Sensu TimescaleDB Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Wavefront integration

The Sensu Wavefront Handler plugin is a Sensu handler that sends Sensu metrics to Wavefront via a proxy, which allows you to store, instrument, and visualize Sensu metrics data in an Wavefront database.

> **NOTE**: *The Sensu Wavefront Handler plugin is an example of Sensu's time-series and long-term event storage integrations. To find more integrations, search Bonsai, the Sensu asset hub.*

Features

- Transform metrics to Wavefront format: extract and transform the metrics you collect from different sources in formats like Graphite, OpenTSDB, Nagios, and Influx and populate them into Wavefront.

- Specify additional tags to include when processing metrics with the Wavefront plugin's `tags` flag or metric tags.

- Keep your Graphite host and port secure with Sensu environment variables and secrets management.

Get the plugin

For a turnkey experience with the Sensu Wavefront Handler plugin, use our curated, configurable quick-start template to integrate Sensu with your existing workflows and store Sensu metrics in Wavefront.

To build your own workflow or integrate Sensu with existing workflows, add the Sensu Wavefront Handler plugin with a dynamic runtime asset from Bonsai, the Sensu asset hub. Dynamic runtime assets are shareable, reusable packages that make it easier to deploy Sensu plugins.

Learn Sensu

The Learn Sensu category includes tools to help you understand and start using Sensu, the industry-leading observability pipeline for multi-cloud monitoring, consolidating monitoring tools, and filling observability gaps at scale.

> **PRO TIP**: *In addition to these learning resources, try the self-guided Sensu Go Workshop.*

Concepts and terminology

If you're new to Sensu, start with a basic review of Sensu concepts and terminology, which includes definitions and links to relevant reference documentation for more in-depth information.

To visualize how Sensu concepts work together in the observability pipeline, take the tour — follow the `Next` buttons on each page.

Live demo

Explore a live demo of the Sensu web UI: view the Entities page to learn what Sensu is monitoring, the Events page for the latest observability events, and the Checks page for active service and metric checks.The live demo also gives you a chance to try commands with sensuctl, the Sensu command line tool.

Sensu sandbox

For further learning opportunities, download the Sensu sandbox.Follow the sandbox lessons to build your first monitoring and observability workflow with Sensu.

We also have a GitHub lesson that guides you through deploying a Sensu cluster and example application into Kubernetes, plus a configuration that allows you to reuse Nagios-style monitoring checks to monitor the example application with a Sensu sidecar.

Sensu concepts and terminology

Agent

A lightweight client that runs on the infrastructure components you want to monitor. Agents self-register with the backend, send keepalive messages, and execute monitoring checks. Each agent belongs to one or more subscriptions that determine which checks the agent runs. An agent can run checks on the entity it's installed on or connect to a remote proxy entity. Read more about the Sensu agent.

Asset

An executable that a check, handler, or mutator can specify as a dependency. Dynamic runtime assets must be a tar archive (optionally gzipped) with scripts or executables within a bin folder. At runtime, the backend or agent installs required assets using the specified URL. Dynamic runtime assets let you manage runtime dependencies without using configuration management tools. Read more about dynamic runtime assets.

Backend

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

Check

A recurring check the agent runs to determine the state of a system component or collect metrics. The backend is responsible for storing check definitions, scheduling checks, and processing observation data (events). Check definitions specify the command to be executed, an interval for execution, one or more subscriptions, and one or more handlers to process the resulting event data. Read more about checks.

Entity

Infrastructure components that you want to monitor.Each entity runs an agent that executes checks and creates events.Events can be tied to the entity where the agent runs or a proxy entity that the agent checks remotely.Read more about entities.

Event

A representation of the state of an infrastructure component at a point in time.The Sensu backend uses events to power the observability pipeline.Observation data in events include the result of a check or metric (or both), the executing agent, and a timestamp.Read more about events.

Event filter

Logical expressions that handlers evaluate before processing observability events.Event filters can instruct handlers to allow or deny matching events based on day, time, namespace, or any attribute in the observation data (event).Read more about event filters.

Handler

A component of the observability pipeline that acts on events.Handlers can send observability data to an executable (or handler plugin), a TCP socket, or a UDP socket.Read more about handlers.

Hook

A command the agent executes in response to a check result *before* creating an observability event. Hooks create context-rich events by gathering relevant information based on check status.Read more about hooks.

Mutator

An executable the backend runs prior to a handler to transform observation data (events).Read more about mutators.

Plugin

Executables designed to work with Sensu observation data (events) either as a check, mutator, or handler plugin. You can write your own check executables in Go, Ruby, Python, and more, or use one of more than 200 plugins shared by the Sensu community. Read more about plugins.

Proxy entities

Components of your infrastructure that can't run the agent locally (like a network switch or a website) but still need to be monitored. Agents create events with information about the proxy entity in place of the local entity when running checks with a specified proxy entity ID. Read more about proxy entities.

Role-based access control (RBAC)

Sensu's local user management system. RBAC lets you manage users and permissions with namespaces, users, roles, and role bindings. Read more about RBAC.

Resources

Objects within Sensu that you can use to specify access permissions in Sensu roles and cluster roles. Resources can be specific to a namespace (like checks and handlers) or cluster-wide (like users and cluster roles). Read more about resources.

Sensuctl

The Sensu command line tool that lets you interact with the backend. You can use sensuctl to create checks, view events, create users, manage clusters, and more. Read more about sensuctl.

Silencing

Entries that allow you to suppress execution of event handlers on an ad-hoc basis. Use silencing to

schedule maintenance without being overloaded with alerts.<u>Read more about silencing</u>.

Subscriptions

Attributes used to indicate which entities will execute which checks.For Sensu to execute a check, the check definition must include a subscription that matches the subscription of at least one Sensu entity. Subscriptions allow you to configure check requests in a one-to-many model for entire groups or subgroups of entities rather than a traditional one-to-one mapping of configured hosts or observability checks.<u>Read more about subscriptions</u>.

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.<u>Read more about tokens</u>.

Live demonstration of Sensu

Try a live demo of the Sensu web UI. Log in with username `guest` and password `i<3sensu`.

Explore the Entities page to learn what Sensu is monitoring, the Events page for the latest observability events, and the Checks page for active service and metric checks.

You can also use the demo to try out sensuctl, the Sensu command line tool. First, install sensuctl on your workstation. Then, configure sensuctl to connect to the demo.

Run `sensuctl configure` and enter the following information:

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

With sensuctl configured, to view the latest observability events, run:

```
sensuctl event list
```

Read the sensuctl documentation to get started using sensuctl.

About the demo

The Caviar project shown in the demo monitors the Sensu docs site using a licensed Sensu cluster of three backends.

Sensu sandbox

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

Learn Sensu in the sandbox

Download the Sensu sandbox and build your first monitoring workflow to send keepalive alerts to Slack. The Learn Sensu sandbox is a CentOS 7 virtual machine pre-installed with Sensu, InfluxDB, and Grafana.

Monitor containers and applications

Follow the instructions for Getting Started with Sensu Go on Kubernetes to deploy a Sensu cluster and an example application (NGINX) into Kubernetes with a Sensu agent sidecar. You'll also learn to use sensuctl to configure a Nagios-style monitoring check for the example application.

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 and learn how to visually inspect and adjust check token substitution and extended attributes to make sure your Core checks work properly in Sensu Go.

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 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 web UIs. Web UI 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 observability event

First, make sure everything is working correctly using the sensuctl command line tool.

Run this command to confirm that your Sensu backend instance has a single namespace, `default` :

```
sensuctl namespace list
```

The response should be:

```
 Name
_____
 default
```

Run this command to list the users for your Sensu backend instance:

```
sensuctl user list
```

The response should list the default admin user and the user created for a Sensu agent to use:

```
 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
```

The response should be an empty 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 confirm that Sensu is now monitoring the sandbox entity:

```
sensuctl entity list
```

The response should include the `sensu-go-sandbox` agent entity:

```
   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 view the keepalive events generated by the sandbox entity:

```
sensuctl event list
```

The response should include the `sensu-go-sandbox` keepalive event:

```
   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 view the event and the entity in the Sensu web UI. Log in to the web UI 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 dynamic runtime asset

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins. In this lesson, we'll use the Sensu Slack handler dynamic runtime asset to power a `slack` handler.

Use sensuctl to register the Sensu Slack handler dynamic runtime 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 receive a confirmation message from sensuctl:

```
Created
```

The `sensu-slack-handler` dynamic runtime asset is now ready to use with Sensu. Use sensuctl to review the complete asset definition.

SHELL

```
sensuctl asset info sensu-slack-handler --format yaml
```

```
sensuctl asset info sensu-slack-handler --format wrapped-json
```

> **PRO TIP**: *You can use resource definitions to create and update resources (like dynamic runtime assets) using* `sensuctl create --file filename.yaml` *or* `sensuctl create --file filename.json` *.Read the* <u>sensuctl docs</u> *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` dynamic runtime asset.

> **NOTE**: *If you aren't sure how to open the handler and edit the definition, try these* <u>Vi/Vim gist instructions</u>.

```
"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 list available event handlers:

```
sensuctl handler list
```

In this case, the response will only list the `keepalive` handler you just created:

```
  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 observability 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, dynamic 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
```

The response should be similar to this example:

```
=== 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 receive 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 <u>InfluxDB</u> and visualize it with <u>Grafana</u>.

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 <u>Sensu HTTP Plugin</u> 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
```

You should receive an OK response:

```
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"
```

The response should end with this line:

```
...
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 dynamic runtime 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 receive a confirmation message from sensuctl:

```
Created
```

The `sensu-influxdb-handler` dynamic runtime asset is now ready to use with Sensu. Use sensuctl to review 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` dynamic runtime 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 in the response. If you completed <u>lesson #2</u>, the response will also list 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
```

Run this command to list the checks:

```
sensuctl check list
```

The response should be similar to:

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 view the event produced by the entity:

```
sensuctl event info sensu-go-sandbox curl_timings --format json | jq .
```

The response should be similar to:

```
...
  "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. Review the HTTP response code events for Nginx in Grafana.

Log in to Grafana with username: `admin` and password: `admin`. When the page loads, it should include a graph of live HTTP response codes for Nginx.

Now, if you turn Nginx off, Grafana should display the effect of the change:

```
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
```

The response should include a list similar to this example:

```
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 can view the working check on the web UI Entity page and via sensuctl:

```
sensuctl event list
```

Now, Grafana should display disk usage metrics for the sandbox: reload your Grafana tab to show the Sensu Go Sandbox Combined.

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 lesson in the Sensu sandbox
- ¬ Install Sensu Go
- ¬ Collect StatsD metrics
- ¬ Create a read-only user