Sensu Go

Star | Share your feedback

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

Sensu Go is the latest version of Sensu, designed to be more portable, easier and faster to deploy, and (even more) friendly to containerized and ephemeral environments.

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

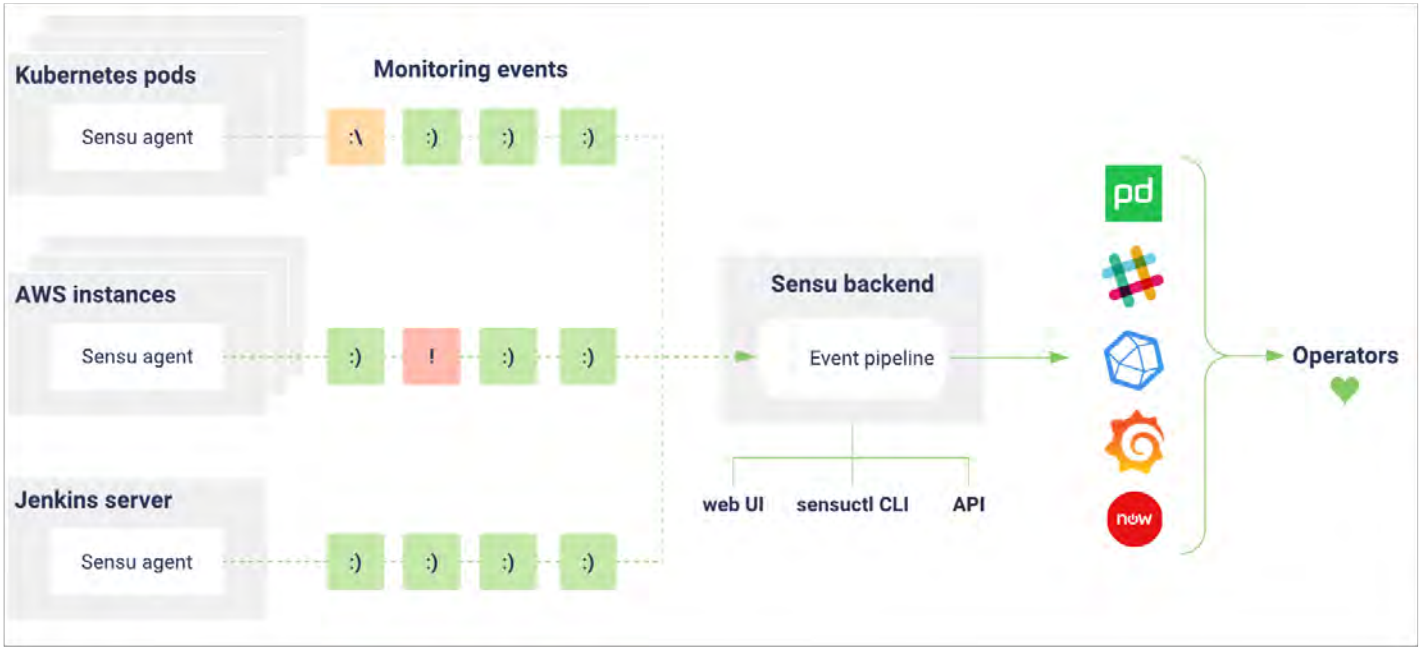
Reduce alert fatigue: Sensu gives you full control over your alerts with flexible filters, context-rich notifications, reporting, event handling, and auto-remediation.

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

Listen to Sensu Inc. CEO Caleb Hailey explain the Sensu monitoring event pipeline.

Monitoring for Your Infrastructure

> Monitoring is the action of observing and checking the behaviors and outputs of a system and its components over time. - Greg Poirier, Monitorama 2016



Sensu is an agent-based monitoring tool that you install on your organization's infrastructure.The Sensu agent gives you visibility into everything you care about; the Sensu server gives you flexible, automated workflows to route metrics and alerts.

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

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

□ Better incident response with filterable, context-rich alerts

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

□ Collect and store metrics with built-in support for industry-standard tools

Know what's going on everywhere in your system.Sensu supports industry-standard metric formats like Nagios Performance Data, Graphite Plaintext Protocol, InfluxDB Line Protocol, OpenTSDB Data Specification, and StatsD metrics.Use the Sensu agent to collect metrics alongside check results, then use the event pipeline to route the data to a time series database like InfluxDB.

□ Intuitive API and dashboard interfaces

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

- **Open core software backed by Sensu Inc.**

Sensu Go's core is open source software, freely available under apermissive MIT License and publicly available on GitHub.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Sensu Go release notes

- 5.0.1 release notes
- 5.0.0 release notes

Versioning

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

Upgrading

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

5.0.1 release notes

December 12, 2018 — Sensu Go 5.0.1 includes our top bug fixes following last week's general availability release.See the upgrade guide to upgrade Sensu to version 5.0.1.

FIXED:

- The Sensu backend can now successfully connect to an external etcd cluster.
- The Sensu dashboard now sorts silencing entries in ascending order, correctly displays status values, and reduces shuffling in the event list.
- Sensu agents on Windows now execute command arguments correctly.

- Sensu agents now correctly include environment variables when executing checks.
- Command arguments are no longer escaped on Windows.
- Sensu backend environments now include handler and mutator execution requests.

5.0.0 release notes

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

For a complete list of changes from Beta 8-1, see the Sensu Go changelog.Going forward, 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

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Getting Started

- Enterprise
- Faq
- Get-Started
- Glossary
- Learn-Sensu
- Media
- Prometheus-Metrics
- Sample-App
- Sandbox

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Getting started with enterprise features

Enterprise features for Sensu Go are available in version 5.2.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest documentation to get started.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Sensu frequently asked questions

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

What platforms does Sensu support?

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

Is Sensu available as a hosted solution?

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

What are the hardware requirements for running a Sensu backed?

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

Is there an enterprise version of Sensu Go?

Yes! Enterprise features for Sensu Go are available in version 5.2.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest documentation to get started.

How can I contact the Sensu sales team?

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

What can I monitor with Sensu?

Sensu supports a wide range of plugins for monitoring everything from the server closet to the cloud. Install the Sensu agent on the hosts you want to monitor, integrate with the Sensu API, or take advantage of proxy entities to monitor anything on your network.Check out the 200+ plugins shared by the Sensu community, including monitoring checks for AWS, Jenkins, Puppet, InfluxDB, and SNMP. You can also get started writing your own Sensu Plugins in any language using the Sensu Plugins spec.

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

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

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

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

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

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

Which ports does Sensu use?

The Sensu backend uses:

- 2379 (HTTP/HTTPS) Sensu storage client: Required for Sensu backends using an external etcd instance
- 2380 (HTTP/HTTPS) Sensu storage peer: Required for other Sensu backends in a cluster
- 3000 (HTTP/HTTPS) Sensu dashboard: Required for all Sensu backends using a Sensu dashboard

- 8080 (HTTP/HTTPS) Sensu API: Required for all users accessing the Sensu API
- 8081 (WS/WSS) Agent API: Required for all Sensu agents connecting to a Sensu backend

The Sensu agent uses:

- 3030 (TCP/UDP) Sensu agent socket: Required for Sensu agents using the agent socket
- 3031 (HTTP) Sensu agent API: Required for all users accessing the agent API
- 8125 (UDP, TCP on Windows) StatsD listener: Required for all Sensu agents using the StatsD listener

For more information, see the guide to securing Sensu.

Can one Sensu backend monitor multiple sites?

Yes, as long as the port requirements described above are met, a single Sensu backend can monitor Sensu agents at multiple sites.

Is it possible to use Uchiwa with Sensu Go?

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

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Get started with Sensu

Try the sandbox

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

- Download the sandbox and learn Sensu Go
- See more sandbox lessons

Install Sensu Go

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

- Install Sensu Go

Create a monitoring workflow

Sensu lets you create automated monitoring workflows to route system metrics and alerts.Get started by following one of the Sensu Go guides.

- Monitor server resources
- Send Slack alerts
- Collect StatsD metrics
- Store metrics with InfluxDB

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Glossary of Terms

Agent

A lightweight client that runs on the infrastructure components you want to monitor.Agents self-register with the backend, send keepalive messages, and execute monitoring checks.Each agent belongs to one or more subscriptions that determine which checks the agent runs.An agent can run checks on the entity it's installed on or by connecting to a remote proxy entity. Read more.

Asset

An asset is an executable that a check, handler, or mutator can specify as a dependency.Assets must be a tar archive (optionally gzipped) with scripts or executables within a bin folder.At runtime, the backend or agent installs required assets using the specified URL.Assets let you manage runtime dependencies without using configuration management tools. Read more.

Backend

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

Check

A recurring check run by the agent to determine the state of a system component or collect metrics. The backend is responsible for storing check definitions, scheduling checks, and processing event data.Check definitions specify the command to be executed, an interval for execution, one or more subscriptions, and one or more handlers to process the resulting event data. Read more.

Check hook

A command executed by the agent in response to a check result, before creating a monitoring event. Hooks create context-rich events by gathering related information based on the check status. Read more.

Check token

A placeholder used 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 re-using the check definition.Read more.

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.

Event

A representation of the state of an infrastructure component at a point in time, used by the backend to power the monitoring event pipeline.Event data includes the result of the check or metric (or both), the executing agent, and a timestamp. Read more.

Filter

Logical expressions that handlers evaluate before processing monitoring events.Filters can instruct handlers to allow or deny matching events based on day, time, namespace, or any attribute in the event data.Read more.

Handler

A component of the monitoring event pipeline that acts on events.Handlers can send monitoring event data to an executable (or handler plugin), a TCP socket, or a UDP socket. Read more.

Mutator

An executable run by the backend prior to the handler to transform event data. Read more.

Plugin

Sensu Plugins are executables designed to work with Sensu event data, either as a check plugin, mutator plugin, or handler plugin.You can write your own check executables in Go, Ruby, Python, and more, or use one of over 200 plugins shared by the Sensu Community. Read more.

Proxy Entity

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.

RBAC

Role-based access control (RBAC) is Sensu's local user management system.RBAC lets you manage users and permissions with namespaces, users, roles, and role bindings.Read more.

Resources

Objects within Sensu that can be used 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.

Sensuctl

Command line tool that lets you interact with the backend.You can use sensuctl to create checks, view events, create users, manage cluster, and more. Read more.

Silencing

Silencing entries allow you to suppress execution of event handlers on an ad-hoc basis.You can use silencing to schedule maintenances without being overloaded with alerts. Read more.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Learn Sensu Go

In this tutorial, we'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
```

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 is intended for use as a learning tool; we do not recommend this tool as part of a production installation.To install Sensu in production, please see the installation guide.The sandbox startup process takes about five minutes.

NOTE: The sandbox configures VirtualBox to forward TCP ports 3002 and 4002 from the sandbox virtual machine to the localhost to make it easier for you to interact with the sandbox dashboards. Dashboard links provided in this tutorial assume port forwarding from the VM to the host is active.

4. SSH into the sandbox

Thanks for waiting! To start using the sandbox:

```
vagrant ssh
```

You should now have shell access to the sandbox and should be greeted with this prompt:

```
[sensu_go_sandbox]$
```

To exit out of the sandbox, use `CTRL` + `D`.To erase and restart the sandbox, use `vagrant destroy` then `vagrant up`.To reset the sandbox's Sensu configuration to the beginning of this tutorial, use `vagrant provision`.

NOTE: The sandbox pre-configures sensuctl with the Sensu Go admin user, so you won't have to configure sensuctl each time you spin up the sandbox to try out a new feature. Before installing sensuctl outside of the sandbox, read the first time setup reference to learn how to configure sensuctl.

Lesson #1: Create a Sensu monitoring event

First off, we'll make sure everything is working correctly by using the sensuctl command line tool.We can use sensuctl to see that our Sensu backend instance has a single namespace, `default`, and two users: the default admin user and the user created for use by a Sensu agent.

```
sensuctl namespace list
 Name
────────
 default

sensuctl user list
 Username    Groups     Enabled
────────── ──────────────── ─────────
admin    cluster-admins  true
agent    system:agents   true
```

Sensu keeps track of monitored components as entities.Let's start by using sensuctl to make sure Sensu hasn't connected to any entities yet:

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

Now we can go ahead and start the Sensu agent to start monitoring the sandbox:

```
sudo systemctl start sensu-agent
```

We can use sensuctl to see that Sensu is now monitoring the sandbox entity:

```
sensuctl entity list
       ID        Class    OS       Subscriptions             Last Seen
────────────────── ─────── ─────── ─────────────────────────
───────────────────────────────
sensu-go-sandbox  agent  linux  entity:sensu-go-sandbox  2019-01-24 21:29:06 +0000 UTC
```

Sensu agents send keepalive events to help you monitor their status.We can use sensuctl to see the keepalive events generated by the sandbox entity:

```
sensuctl event list
    Entity        Check                          Output                          Status   Silenced          Timestamp
──────────────── ──────────
─────────────────────────────────────────────────────────────────────
─────── ───────── ────────────────────────────
sensu-go-sandbox   keepalive   Keepalive last sent from sensu-go-sandbox at 2019-01-24 21:29:06 +0000 UTC        0
false      2019-01-24 21:29:06 +0000 UTC
```

The sensu-go-sandbox keepalive event has status 0, meaning the agent is in an OK state and able to communicate with the Sensu backend.

We can also see the event and the entity in the Sensu dashboard.Log in to the dashboard as the default admin user: username `admin` and password `P@ssw0rd!`.

Lesson #2: Pipe keepalive events into Slack

Now that we know the sandbox is working properly, let's get to the fun stuff: creating a workflow.In this lesson, we'll create a workflow that sends keepalive alerts to Slack.(If you'd rather not create a Slack account, you can skip ahead to lesson 3.)

1. Get your Slack webhook URL

If you're already an 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, start here to create a new workspace.)After saving, you'll see your webhook URL under Integration Settings.

2. Register the Sensu Slack handler asset

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins.In this lesson, we'll use the Sensu Slack handler asset to power a `slack` handler.

Use sensuctl to register the Sensu Slack handler asset.

```
sensuctl asset create sensu-slack-handler --url "https://github.com/sensu/sensu-sla
handler_1.0.3_linux_amd64.tar.gz" --sha512
```

```
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b5871
```

You should see a confirmation message from sensuctl.

```
Created
```

The `sensu-slack-handler` asset is now ready to use with Sensu.You can use sensuctl to see the complete asset definition.

```
sensuctl asset info sensu-slack-handler --format yaml
```

PRO TIP: You can use resources definition to create and update resources (like assets) using `sensuctl create --file filename.yaml`. See the sensuctl docs for more information.

3. Create a Sensu Slack handler

Open the `sensu-slack-handler.json` handler definition provided with the sandbox, and edit the definition to include your Slack channel, webhook URL, and the `sensu-slack-handler` asset.

```
"env_vars": [
  "KEEPALIVE_SLACK_WEBHOOK=https://hooks.slack.com/services/AAA/BBB/CCC",
  "KEEPALIVE_SLACK_CHANNEL=#monitoring"
],
"runtime_assets": ["sensu-slack-handler"]
```

Now we can create a Slack handler named `keepalive` to process keepalive events.

```
sensuctl create --file sensu-slack-handler.json
```

You can use sensuctl to see available event handlers.

```
sensuctl handler list
```

You should see the `keepalive` handler.

```
  Name      Type   Timeout   Filters   Mutator                                   Execute
Environment Variables                         Assets
 ─────────── ────── ───────── ───────── ─────────
─────────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────────────────────────────────────
─────────────────────
 keepalive   pipe        0                  RUN:   /usr/local/bin/sensu-slack-handler -c "${KEEPALIVE_SLACK_CHANNEL}"
"${KEEPALIVE_SLACK_WEBHOOK}"
KEEPALIVE_SLACK_WEBHOOK=https://hooks.slack.com/services/XXX,KEEPALIVE_SLACK_CHANNEL=#monitor
```

You should now see monitoring events in Slack indicating that the sandbox entity is in an OK state.

4. Filter keepalive events

Now that we're generating Slack alerts, let's reduce the potential for alert fatigue by adding a filter that only sends only warning, critical, and resolution alerts to Slack.

To accomplish this, we'll interactively add the built-in is_incident filter to the keepalive handler so we'll only receive alerts when the sandbox entity fails to send a keepalive event.

```
sensuctl handler update keepalive
```

When prompted for the filters selection, enter `is_incident` to apply the incidents filter.

```
? Filters: [? for help] is_incident
```

We can confirm that the keepalive handler now includes the incidents filter using sensuctl:

```
sensuctl handler info keepalive
=== keepalive
```

```
Name:                   keepalive
Type:                   pipe
Timeout:                0
Filters:                is_incident
```

With the filter in place we should no longer be receiving messages in the Slack channel every time the sandbox entity sends a keepalive event.

Let's stop the agent and confirm that we receive the expected warning message.

```
sudo systemctl stop sensu-agent
```

You should see the warning message in Slack after a couple of minutes, informing you that the sandbox entity is no longer sending keepalive events.

Before we go, start the agent to resolve the warning.

```
sudo systemctl start sensu-agent
```

Lesson #3: Automate event production with the Sensu agent

So far we've used the Sensu agent's built-in keepalive feature, but in this lesson, we'll create a check that automatically produces workload-related events.Instead of sending alerts to Slack, we'll store event data with InfluxDB and visualize it with Grafana.

1. Make sure the Sensu agent is running

```
sudo systemctl restart sensu-agent
```

2. Install Nginx and the Sensu HTTP Plugin

We'll use the Sensu HTTP Plugin to monitor an Nginx server running on the sandbox.

First, install and start Nginx:

```
sudo yum install -y nginx && sudo systemctl start nginx
```

And make sure it's working with:

```
curl -I http://localhost:80

HTTP/1.1 200 OK
```

Then install the Sensu HTTP Plugin:

```
sudo sensu-install -p sensu-plugins-http
```

We'll be using the `metrics-curl.rb` plugin.We can test its output using:

```
/opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -u "http://localhost"

...
sensu-go-sandbox.curl_timings.http_code 200 1535670975
```

3. Create an InfluxDB pipeline Now let's create the InfluxDB pipeline to store these metrics and visualize them with Grafana.To create a pipeline to send metric events to InfluxDB, start by registering the Sensu InfluxDB handler asset.

```
sensuctl asset create sensu-influxdb-handler --url "https://github.com/sensu/sensu-i
influxdb-handler_3.1.2_linux_amd64.tar.gz" --sha512
"612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9
```

You should see a confirmation message from sensuctl.

```
Created
```

The `sensu-influxdb-handler` asset is now ready to use with Sensu.You can use sensuctl to see the complete asset definition.

```
sensuctl asset info sensu-influxdb-handler --format yaml
```

Open the `influx-handler.json` handler definition provided with the sandbox, and edit the `runtime_assets` attribute to include the `sensu-influxdb-handler` asset.

```
"runtime_assets": ["sensu-influxdb-handler"]
```

Now you can use sensuctl to create the `influx-db` handler.

```
sensuctl create --file influx-handler.json
```

We can use sensuctl to confirm that the handler has been created successfully.

```
sensuctl handler list
```

You should see the `influx-db` handler.(If you've completed lesson #2, you'll also see the `keepalive` handler.)

4. Create a check to monitor Nginx

Use the `curl_timings-check.json` file provided with the sandbox to create a service check that runs `metrics-curl.rb` every 10 seconds on all entities with the `entity:sensu-go-sandbox` subscription and sends events to the InfluxDB pipeline:

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

sensuctl check list
```

```
    Name                          Command                         Interval   Cron   Timeout   TTL      Subscriptions
Handlers   Assets   Hooks   Publish?   Stdin?    Metric Format     Metric Handlers
────────────────
 ───────────────────────────────────────────────────────────────────────────────────
 ──────────── ─────── ────────── ────── ─────────────────────────── ──────────── ──────────
 ──────── ──────────── ───────── ────────────────────── ───────────────────
curl_timings   /opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -u "http://localhost"       10             0    0
entity:sensu-go-sandbox                              true       false    graphite_plaintext   influx-db
```

This check defines a metrics handler and metric format.In Sensu Go metrics are a core element of the data model, so we can build pipelines to handle metrics separately from alerts.This allows us to customize our monitoring workflows to get better visibility and reduce alert fatigue.

After about 10 seconds, we can see the event produced by the entity:

```
sensuctl event info sensu-go-sandbox curl_timings --format json | jq .
...
  "metrics": {
    "handlers": [
      "influx-db"
    ],
    "points": [
      {
        "name": "sensu-go-sandbox.curl_timings.time_total",
        "value": 0.005,
        "timestamp": 1543532948,
        "tags": []
      },
      {
        "name": "sensu-go-sandbox.curl_timings.time_namelookup",
        "value": 0.005,
        "timestamp": 1543532948,
        "tags": []
      },
      {
        "name": "sensu-go-sandbox.curl_timings.time_connect",
        "value": 0.005,
        "timestamp": 1543532948,
        "tags": []
```

```
      }
    ]
  }
```

Because we configured a metric format, the Sensu agent was able to convert the Graphite-formatted metrics provided by the check command into a set of Sensu-formatted metrics.Metric support isn't limited to just Graphite; the Sensu agent can extract metrics in multiple line protocol formats, including Nagios performance data..

5. See the HTTP response code events for Nginx in Grafana**.**

Log in to Grafana as username: `admin` and password: `admin`.We should see a graph of live HTTP response codes for Nginx.

Now if we turn Nginx off, we should see the impact in Grafana:

```
sudo systemctl stop nginx
```

Start Nginx:

```
sudo systemctl start nginx
```

6. Automate disk usage monitoring for the sandbox

Now that we have an entity set up, we can easily add more checks.For example, let's say we want to monitor disk usage on the sandbox.

First, install the plugin:

```
sudo sensu-install -p sensu-plugins-disk-checks
```

And test it:

```
/opt/sensu-plugins-ruby/embedded/bin/metrics-disk-usage.rb
```

```
sensu-core-sandbox.disk_usage.root.used 2235 1534191189
sensu-core-sandbox.disk_usage.root.avail 39714 1534191189
...
```

Then create the check using sensuctl and the `disk_usage-check.json` file included with the sandbox, assigning it to the `entity:sensu-go-sandbox` subscription and the InfluxDB pipeline:

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

We should see it working in the dashboard entity view and via sensuctl:

```
sensuctl event list
```

Now we should be able to see disk usage metrics for the sandbox in Grafana.

You made it! You're ready for the next level of Sensu-ing.Here are some resources to help continue your journey:

- Install Sensu Go
- Collect StatsD metrics
- Create a ready-only user

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Sensu Go media

Talks

- Greg Poirier - Sensu Go Deep Dive at Sensu Summit 2017
- Greg Poirier - Sensu Go Assets
- Sean Porter, Influx Days - Data Collection & Prometheus Scraping with Sensu 5.0

Blog posts

- Simon Plourde: Understanding RBAC in Sensu Go
- Sean Porter: Self-service monitoring checks in Sensu Go
- Christian Michel - How to monitor 1,000 network devices using Sensu Go and Ansible
- Eric Chlebek - Filters: valves for the Sensu monitoring event pipeline
- Greg Schofield - Sensu Habitat Core Plans are Here
- Nikki Attea - Check output metric extraction with InfluxDB & Grafana
- Jef Spaleta - Migrating to 5.0
- Anna Plotkin - Sensu Go is here!

Tutorials

- Sensu sandbox tutorials

Podcasts

- Sensu Community Chat November 2018

NOTE: Prior to October 2018, Sensu Go was known as Sensu 2.0.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Using the Sensu Prometheus Collector

What is the Sensu Prometheus Collector?

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.

Why use Sensu with Prometheus?

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 which Sensu can collect and route to one or more time series databases, such as InfluxDB or Graphite. Both Sensu and Prometheus can run in parallel, complimenting each other and making use of environments where Prometheus is already deployed.

In this guide

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, you will have Prometheus scraping metrics. The Sensu Prometheus Collector will then query the Prometheus API as a Sensu check, send those to an InfluxDB Sensu handler, which will send metrics to an InfluxDB instance. Finally, Grafana will query InfluxDB to display those collected metrics.

Set up

Install and configure Prometheus

Download and extract Prometheus.

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.6.0/prometheus-
2.6.0.linux-amd64.tar.gz

tar xvfz prometheus-*.tar.gz

cd prometheus-*
```

Replace the default `prometheus.yml` configuration file with the following configuration.

```
global:
  scrape_interval: 15s
  external_labels:
    monitor: 'codelab-monitor'

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
```

Start Prometheus in the background.

```
nohup ./prometheus --config.file=prometheus.yml > prometheus.log 2>&1 &
[1] 7647
```

Ensure Prometheus is running.

```
ps -ef | grep prometheus
vagrant   7647  3937  2 22:23 pts/0    00:00:00 ./prometheus --
config.file=prometheus.yml
```

Install and configure Sensu Go

Follow the RHEL/CentOS install instructions for the Sensu backend, Sensu agent and sensuctl.

Add an `app_tier` subscription to `/etc/sensu/agent.yml`.

```
subscriptions:
  - "app_tier"
```

Restart the sensu agent to apply the configuration change.

```
systemctl restart sensu-agent
```

Ensure Sensu services are running.

```
systemctl status sensu-backend
systemctl status sensu-agent
```

Install and configure InfluxDB

Add InfluxDB repo.

```
echo "[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key" | sudo tee
/etc/yum.repos.d/influxdb.repo
```

Install InfluxDB.

```
sudo yum -y install influxdb
```

Open `/etc/influxdb/influxdb.conf` and uncomment the `http` API line.

```
[http]
  # Determines whether HTTP endpoint is enabled.
  enabled = true
```

Start InfluxDB.

```
sudo systemctl start influxdb
```

Add the Sensu user and database.

```
influx -execute "CREATE DATABASE sensu"

influx -execute "CREATE USER sensu WITH PASSWORD 'sensu'"

influx -execute "GRANT ALL ON sensu TO sensu"
```

Install and configure Grafana

Install Grafana.

```
sudo yum install -y https://s3-us-west-2.amazonaws.com/grafana-
releases/release/grafana-5.1.4-1.x86_64.rpm
```

Change Grafana's listen port to not conflict with the Sensu Dashboard.

```
sudo sed -i 's/^;http_port = 3000/http_port = 4000/' /etc/grafana/grafana.ini
```

Create a `/etc/grafana/provisioning/datasources/influxdb.yaml` file, and add an InfluxDB data source.

```
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.

```
systemctl start grafana-server
```

Create a Sensu InfluxDB pipeline

Install Sensu InfluxDB handler

```
wget -q -nc https://github.com/sensu/sensu-influxdb-handler/releases/download/3.0.1/sensu-influxdb-handler_3.0.1_linux_amd64.tar.gz -P /tmp/

tar xvfz /tmp/sensu-influxdb-handler_3.0.1_linux_amd64.tar.gz -C /tmp/

cp /tmp/bin/sensu-influxdb-handler /usr/local/bin/
```

Create a Sensu handler

Given the following handler definition in a file called `handler.json`:

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

```
  "metadata": {
    "name": "influxdb",
    "namespace": "default"
  },
  "spec": {
    "command": "/usr/local/bin/sensu-influxdb-handler -a 'http://127.0.0.1:8086'
-d sensu -u sensu -p sensu",
    "env_vars": [],
    "timeout": 10,
    "type": "pipe"
  }
}
```

Use `sensuctl` to add the handler to Sensu.

```
sensuctl create --file handler.json
```

Collect Prometheus metrics with Sensu

Install Sensu Prometheus Collector

```
wget -q -nc https://github.com/sensu/sensu-prometheus-
collector/releases/download/1.1.4/sensu-prometheus-
collector_1.1.4_linux_386.tar.gz -P /tmp/

tar xvfz /tmp/sensu-prometheus-collector_1.1.4_linux_386.tar.gz -C /tmp/

cp /tmp/bin/sensu-prometheus-collector /usr/local/bin/
```

Confirm the collector can get metrics from Prometheus.

```
/usr/local/bin/sensu-prometheus-collector -prom-url http://localhost:9090 -prom-
query up
```

```
up,instance=localhost:9090,job=prometheus value=1 1549991087
```

Add a Sensu check to complete the pipeline

Given the following check definition in a file called `check.json`:

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "prometheus_metrics",
    "namespace": "default"
  },
  "spec": {
    "command": "/usr/local/bin/sensu-prometheus-collector -prom-url
http://localhost:9090 -prom-query up",
    "handlers": [
      "influxdb"
    ],
    "interval": 10,
    "publish": true,
    "output_metric_format": "influxdb_line",
    "output_metric_handlers": [],
    "subscriptions": [
      "app_tier"
    ],
    "timeout": 0
  }
}
```

PRO TIP: `sensuctl create -f` also accepts files containing multiple resources definitions.

Use `sensuctl` to add the check to Sensu.

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

We can see the events generated by the `prometheus_metrics` check in the Sensu dashboard.Visit http://127.0.0.1:3000 , and log in as the default admin user: username `admin` and password `P@ssw0rd!` .

We can also see the metric event data using sensuctl.

```
sensuctl event list
    Entity         Check                              Output                              Status   Silenced         Timestamp
────────────── ────────────────────

──────────────────────────────────────────────────────────────────────

──────── ────────── ──────────────────────────────
sensu-centos   keepalive          Keepalive last sent from sensu-centos at 2019-02-12 01:01:37 +0000 UTC       0   false
2019-02-12 01:01:37 +0000 UTC
sensu-centos   prometheus_metrics   up,instance=localhost:9090,job=prometheus value=1 1549933306              0
false      2019-02-12 01:01:46 +0000 UTC
```

Visualize metrics with Grafana

Configure a dashboard in Grafana

Download the Grafana dashboard configuration file from the Sensu docs.

```
wget https://docs.sensu.io/sensu-go/5.0/files/up_or_down_dashboard.json
```

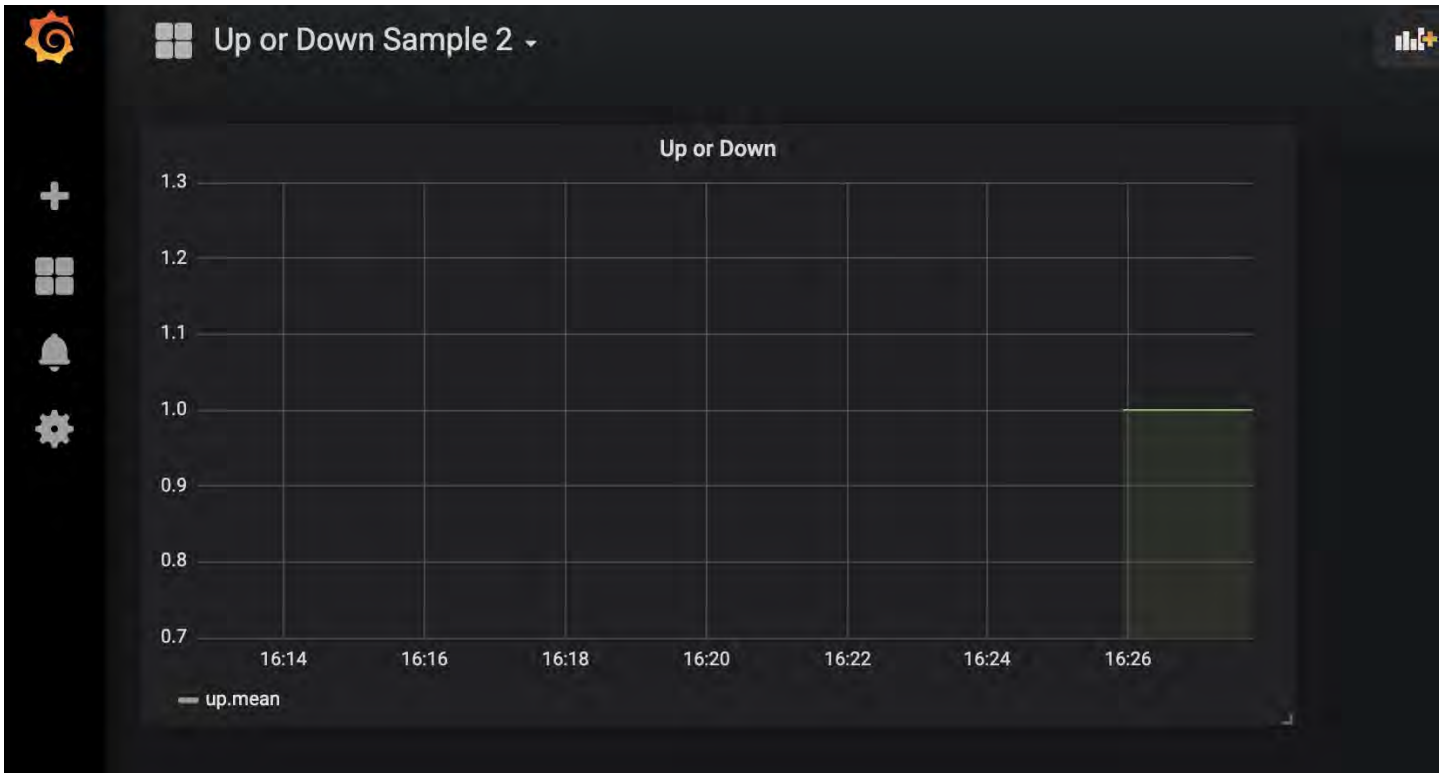
Using the downloaded file, add the dashboard to Grafana using an API call.

```
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 with `admin:admin` login at http://127.0.0.1:4000 .

Once logged in, click on Home in the upper left corner, then below click on the Up or Down Sample 2 dashboard. Once there, you should see a graph that has started showing metrics like this



Conclusion

You should now have a working setup with Prometheus scraping metrics. The Sensu Prometheus Collecting is being ran via a Sensu check and collecting those metrics from Prometheus' API. The metrics are then handled by the InfluxDB handler, sent to InfluxDB and then visualized by a Grafana Dashboard.

Using this information, you can now plug the Sensu Prometheus Collector into your Sensu ecosystem and leverage Prometheus to gather metrics and Sensu to send them to the proper final destination. Prometheus has a comprehensive list of additional exporters to pull in metrics.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Container and application monitoring with Sensu

All Platforms

In this tutorial, we'll deploy a sample app with Kubernetes and monitor it with Sensu.The sample app has three endpoints: `/` returns the local hostname, `/metrics` returns Prometheus metric data, `/healthz` returns the boolean health state, and `POST /healthz` toggles the health state.

Prerequisites

The sample app requires Kubernetes and a Kubernetes Ingress controller.Most hosted Kubernetes offerings, such as GKE, include a Kubernetes Ingress controller.

In this tutorial, we'll be using Minikube, a cross-platform application for running a local single-node Kubernetes cluster.After you've installed and started Minikube, proceed through the rest of the guide.

Setup

1. Clone the sample app.

```
git clone https://github.com/sensu/sensu-kube-demo && cd sensu-kube-demo
```

2. Create the Kubernetes ingress resources.

```
minikube start

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/master/deploy/mandatory.yaml

minikube addons enable ingress

kubectl create -f go/ingress-nginx/ingress/sensu-go.yaml
```

3. Deploy kube-state-metrics.

```
git clone https://github.com/kubernetes/kube-state-metrics

kubectl apply -f kube-state-metrics/kubernetes
```

4. Open your `/etc/hosts` file and add the following hostnames.

NOTE: Here we'll use the IP address for the Minikube VM in our hosts file. To view the address, use the command `minikube ip`.

```
192.168.99.100       sensu.local webui.sensu.local sensu-enterprise.local
dashboard.sensu-enterprise.local
192.168.99.100       influxdb.local grafana.local dummy.local
```

5. Install sensuctl.

Jump over to the sensuctl installation guide, and follow the instructions to install sensuctl on Windows, macOS, or Linux.

6. Deploy two instances of the sample app (dummy) behind a load balancer.

```
kubectl apply -f go/deploy/dummy.yaml
```

We can test the dummy app using the API.

```
# Linux/macOS
curl -i http://dummy.local
```

```
# Windows
Invoke-WebRequest -Uri http://dummy.local -Method GET
```

A `200` response indicates that the dummy app is working correctly.

7. Deploy the Sensu backend

```
kubectl create -f go/deploy/sensu-backend.yaml
```

Multitenancy

Use Sensu role-based access control to create a `demo` namespace and a `demo` user.

1. Configure sensuctl to use the built-in `admin` user.

```
sensuctl configure
? Sensu Backend URL: http://sensu.local
? Username: admin
? Password: P@ssw0rd!
? Namespace: default
? Preferred output format: tabular
```

2. Create a `demo` namespace.

```
sensuctl namespace create demo
```

We can use sensuctl to confirm that the namespace was created successfully and set the `demo` namespace as the default for our sensuctl session.

```
sensuctl namespace list

sensuctl config set-namespace demo
```

3. Create a `dev` user role with full-access to the `demo` namespace.

```
sensuctl role create dev \
--verb get,list,create,update,delete \
--resource \* --namespace demo
```

4. Create a `dev` role binding for the `dev` group.

```
sensuctl role-binding create dev --role dev --group dev
```

5. Create a `demo` user that is a member of the `dev` group.

```
sensuctl user create demo --interactive
? Username: demo
? Password: password
? Groups: dev
```

6. Reconfigure sensuctl to use the `demo` user and `demo` namespace.

```
sensuctl configure
? Sensu Backend URL: http://sensu.local
? Username: demo
? Password: password
? Namespace: demo
? Preferred output format: tabular
```

Deploying Sensu agents and InfluxDB

1. Deploy InfluxDB with a Sensu agent sidecar

Create a Kubernetes ConfigMap for InfluxDB configuration.

```
kubectl create configmap influxdb-config --from-file go/configmaps/influxdb.conf
```

Deploy InfluxDB with a Sensu agent sidecar.

```
kubectl create -f go/deploy/influxdb.sensu.yaml
```

2. Create a Sensu pipeline to store metrics with InfluxDB.

Use the files provided with the sample app to create a Sensu asset for the Sensu InfluxDB handler and create an `influxdb` event handler.

```
sensuctl create --file go/config/assets/influxdb-handler.yaml

sensuctl create --file go/config/handlers/influxdb.yaml
```

3. Deploy Sensu agent sidecars for the dummy app instances.

```
kubectl apply -f go/deploy/dummy.sensu.yaml
```

Monitoring an app

Let's take a look at what we're monitoring.We can see the Sensu agents installed on our two dummy app instances with their last seen timestamp, as well as the Sensu agent monitoring our InfluxDB instance.

```
sensuctl entity list

        ID             Class    OS              Subscriptions                   Last Seen
──────────────────────────────── ──────── ────────
─────────────────────────────────────────────────── ─────────────────────────────────────
dummy-76d8fb7bdf-967q7     agent   linux   dummy,entity:dummy-76d8fb7bdf-967q7        2019-01-18 10:56:56 -0800
PST
dummy-76d8fb7bdf-knh7r     agent   linux   dummy,entity:dummy-76d8fb7bdf-knh7r        2019-01-18 10:56:56 -0800
PST
influxdb-64b7d5f884-f9ptg   agent   linux   influxdb,entity:influxdb-64b7d5f884-f9ptg   2019-01-18 10:56:59 -0800
PST
```

Create a Sensu pipeline to Slack

Let's say we want to receive a Slack alert if the dummy app returns an unhealthy response.We can create a Sensu pipeline to send events to Slack using the Sensu Slack plugin.Sensu Plugins are open-source collections of Sensu building blocks shared by the Sensu Community.

1. Create an asset to help agents find and install the Sensu Slack handler.

```
sensuctl create --file go/config/assets/slack-handler.yaml
```

2. Get your Slack webhook URL and add it to `go/config/handlers/slack.yaml`.

If you're already an 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 and save the settings.(If you're not yet a Slack admin, start here to create a new workspace.)After saving, you'll see your webhook URL under Integration Settings.

Open `go/config/handlers/slack.yaml` and replace `SECRET` in the following line with your Slack workspace webhook URL and `#demo` with the Slack channel of your choice:

```
"command": "slack-handler --channel '#demo' --timeout 20 --username 'sensu' --
webhook-url 'SECRET'",
```

So it looks something like:

```
"command": "slack-handler --channel '#my-channel' --timeout 20 --username
'sensu' --webhook-url 'https://hooks.slack.com/services/XXXXXXXXXXXXXXXX'",
```

3. Create a handler to send events to Slack using the `slack-handler` asset.

```
sensuctl create --file go/config/handlers/slack.yaml
```

Create a Sensu service check to monitor the status of the dummy app

To automatically monitor the status of the dummy app, we'll create an asset that lets the Sensu agents use a Sensu HTTP plugin.

1. Create the `check-plugins` asset.

```
sensuctl create --file go/config/assets/check-plugins.yaml
```

2. Now we can create a check to monitor the status of the dummy app that uses the `check-plugins` asset and the Slack pipeline.

```
sensuctl create --file go/config/checks/dummy-app-healthz.yaml
```

3. With the automated alert workflow in place, we can see the resulting events in the Sensu dashboard.

Sign in to the Sensu dashboard with your sensuctl username (`demo`) and password (`password`). Since we're working within the `demo` namespace, select the `demo` namespace in the Sensu dashboard menu.

4. Toggle the health of the dummy app to simulate a failure.

```
# Linux/macOS
curl -iXPOST http://dummy.local/healthz
```

```
# Windows
Invoke-WebRequest -Uri http://dummy.local/healthz -Method POST
```

We should now be able to see a critical alert in the Sensu dashboard as well as by using sensuctl:

```
sensuctl event list
```

You should also see an alert in Slack.

Continue to post to `/healthz` until all Sensu entities return to a healthy state.

```
# Linux/macOS
curl -iXPOST http://dummy.local/healthz
```

```
# Windows
Invoke-WebRequest -Uri http://dummy.local/healthz -Method POST
```

Collecting app metrics

Create a Sensu metric check to collect Prometheus metrics

To automatically collect Prometheus metrics from the dummy app, we'll create an asset that lets the Sensu agents use the Sensu Prometheus plugin.

1. Create the `prometheus-collector` asset.

```
sensuctl create --file go/config/assets/prometheus-collector.yaml
```

2. Now we can create a check to collect Prometheus metrics that uses the `prometheus-collector` asset.

```
sensuctl create --file go/config/checks/dummy-app-prometheus.yaml
```

Visualize metrics with Grafana

1. Deploy Grafana with a Sensu agent sidecar.

Create Kubernetes ConfigMaps for Grafana configuration.

```
kubectl create configmap grafana-provisioning-datasources --from-
file=./go/configmaps/grafana-provisioning-datasources.yaml

kubectl create configmap grafana-provisioning-dashboards --from-
```

```
file=./go/configmaps/grafana-provisioning-dashboards.yaml
```

Deploy Grafana with a Sensu agent sidecar.

```
kubectl apply -f go/deploy/grafana.sensu.yaml
```

After a few minutes, we can see the Sensu agents we have installed on the dummy app, InfluxDB, and Grafana pods.

```
sensuctl entity list
          ID             Class    OS               Subscriptions                    Last Seen
───────────────────────────── ─────── ───────
────────────────────────────────────────────── ─────────────────────────────────
dummy-6c57b8f868-ft5dz     agent   linux   dummy,entity:dummy-6c57b8f868-ft5dz        2018-11-20 18:43:15 -0800 PST
dummy-6c57b8f868-m24hw     agent   linux   dummy,entity:dummy-6c57b8f868-m24hw        2018-11-20 18:43:15 -0800 PST
grafana-5b88f8df8d-vgjtm    agent   linux   grafana,entity:grafana-5b88f8df8d-vgjtm     2018-11-20 18:43:14 -0800 PST
influxdb-78d64bcfd9-8km56   agent   linux   influxdb,entity:influxdb-78d64bcfd9-8km56   2018-11-20 18:43:12 -0800 PST
```

2. Log in to Grafana.

To see the metrics we're collecting from the dummy app, log into Grafana with the username `admin` and password `password`.

3. Create a dashboard.

Create a new dashboard using the InfluxDB datasource to see live metrics from the dummy app.

Collecting Kubernetes metrics

Now that we have a pipeline set up to send metrics, we can create a check that collects Prometheus metrics from Kubernetes and connect it to the pipeline.

Deploy a Sensu agent as a dameonset on your Kubernetes node.

```
kubectl apply -f go/deploy/sensu-agent-daemonset.yaml
```

Then create a check to collect Prometheus metrics from Kubernetes using the `prometheus-collector` asset and `influxdb` handler.

```
sensuctl create --file go/config/checks/kube-state-prometheus.yaml
```

You should now be able to access Kubernetes metric data in Grafana and see metric events in the Sensu dashboard.

Next steps

To stop or delete the sample app, use `minikube stop` or `minikube delete` respectively.

For more information about monitoring with Sensu, check out the following resources:

- Reducing alert fatigue with Sensu filters
- Aggregating StatD metrics with Sensu
- Aggregating Nagios metrics with Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Sensu sandbox

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

Learn Sensu

- Start here: Building your first monitoring workflow

Container monitoring

- Container and application monitoring with Sensu: Monitoring a Kubernetes sample app

Metrics

- Sensu + Prometheus: Collecting Prometheus metrics with Sensu

Upgrading from Sensu 1.x to Sensu Go

- Sensu translator: Translating check configuration

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Installation

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Authentication

Sensu requires username and password authentication to access the Sensu dashboard, API, and command line tool (sensuctl).For Sensu's default user credentials, and more information about configuring Sensu role based access control, see the RBAC reference and guide to creating users.

In addition to built-in RBAC, enterprise-only support for authentication using an authentication provider is available in Sensu Go 5.2.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest documentation to configure an authentication provider.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Configuration Management

We highly 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 Servers.

The following configuration management tools have well-defined Sensu modules to help you get started.

Puppet

The Puppet Sensu module can be found on the GitHub.Sensu has partnered with Learn Puppet to enhance the Puppet module with new features and bug fixes.

Chef

The Chef cookbook for Sensu can be found on the GitHub. Interested in more information on Sensu + Chef? Get some helpful resources here.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Installing Sensu

All Platforms

Select a platform from the dropdown above.Sensu Go is available for Linux, Windows (agent and CLI only), macOS (CLI only), and Docker.See the list of supported platforms for more information.Sensu downloads are provided under the Sensu License.

Install the Sensu backend

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

1. Install the package

Ubuntu

Add the Sensu repository.

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh
| sudo bash
```

Install the `sensu-go-backend` package.

```
sudo apt-get install sensu-go-backend
```

RHEL/CentOS

Add the Sensu repository.

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh
| sudo bash
```

Install the `sensu-go-backend` package.

```
sudo yum install sensu-go-backend
```

2. Create the configuration file

Copy the example backend config file to the default config path.

```
sudo cp /usr/share/doc/sensu-go-backend-5.0.1/backend.yml.example
/etc/sensu/backend.yml
```

NOTE: The Sensu backend can be configured using a `/etc/sensu/backend.yml` configuration file or using `sensu-backend start` configuration flags. For more information, see the backend reference.

3. Start the service

Start the backend using a service manager.

```
sudo service sensu-backend start
```

Verify that the backend is running.

```
service sensu-backend status
```

Next steps

Now that you've installed the Sensu backend:

- [ ] Install the Sensu agent
- [ ] Install sensuctl
- [ ] Sign in to the dashboard

Install the Sensu agent

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

1. Install the package

Ubuntu

Add the Sensu repository.

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh
| sudo bash
```

Install the `sensu-go-agent` package.

```
sudo apt-get install sensu-go-agent
```

RHEL/CentOS

Add the Sensu repository.

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh
| sudo bash
```

Install the `sensu-go-agent` package.

```
sudo yum install sensu-go-agent
```

Windows

Download the Sensu agent for Windows.

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-
go/5.0.1/sensu-go-5.0.1-windows-amd64.tar.gz  -OutFile "$env:userprofile\sensu-
go-5.0.1-windows-amd64.tar.gz"
```

See the verifying Sensu guide to verify your download using checksums.

2. Create the configuration file

Ubuntu/RHEL/CentOS

Copy the example agent config file to the default config path.

```
sudo cp /usr/share/doc/sensu-go-agent-5.0.1/agent.yml.example
/etc/sensu/agent.yml
```

NOTE: The Sensu agent can be configured using a `/etc/sensu/agent.yml` configuration file or using `sensu-agent start` configuration flags. For more information, see the agent reference.

Windows

Download the example agent configuration file and save it as `C:\\ProgramData\sensu\config\agent.yml`.

3. Start the service

Ubuntu/RHEL/CentOS

Start the agent using a service manager.

```
sudo service sensu-agent start
```

Verify that the agent is running.

```
service sensu-agent status
```

Windows

Coming soon.

Next steps

Now that you've installed the Sensu agent:

- Install sensuctl
- Create a monitoring event

Install sensuctl

Sensu Go can be configured and used with the sensuctl command line utility.Sensuctl is available for

Ubuntu, RHEL/CentOS, Windows, and macOS.

1. Install the package

Ubuntu

Add the Sensu repository.

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.deb.sh
| sudo bash
```

Install the `sensu-go-cli` package.

```
sudo apt-get install sensu-go-cli
```

RHEL/CentOS

Add the Sensu repository.

```
curl -s https://packagecloud.io/install/repositories/sensu/stable/script.rpm.sh
| sudo bash
```

Install the `sensu-go-cli` package.

```
sudo yum install sensu-go-cli
```

Windows

Download sensuctl for Windows.

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.0.1/sensu-go-5.0.1-windows-amd64.tar.gz  -OutFile C:\Users\Administrator\sensu-go-5.0.1-windows-amd64.tar.gz
```

See the verifying Sensu guide to verify your download using checksums.

macOS

Download the latest release. See the verifying Sensu guide to verify your download using checksums.

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.0.1/sensu-go-5.0.1-darwin-amd64.tar.gz
```

Extract the archive.

```
tar -xvf sensu-go-5.0.1-darwin-amd64.tar.gz
```

Copy the executable into your PATH.

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

2. Configure sensuctl

You must configure sensuctl before it can connect to Sensu Go.Run `sensuctl configure` to get started.

```
$ sensuctl configure
? Sensu Backend URL: http://127.0.0.1:8080
? Username: admin
? Password: *********
? Namespace: default
```

```
? Preferred output format: tabular
```

By default, your Sensu installation comes with a user named `admin` with password `P@ssw0rd!`.We **strongly** recommended that you change the password immediately.Once authenticated, you can change the password using the `change-password` command.

```
$ sensuctl user change-password --interactive
? Current Password:  *********
? Password:          *********
? Confirm:           *********
```

You can change individual values of your sensuctl configuration with the `config` subcommand.

```
sensuctl config set-namespace default
```

See the sensuctl reference for more information about using sensuctl.

Next steps

Now that you've installed sensuctl:

- See the sensuctl quick reference
- Create a monitoring event pipeline

Deploy Sensu with Docker

Sensu Go can be run via Docker or rkt using the sensu/sensu image. When running Sensu from Docker there are a couple of things to take into consideration.

The backend requires four exposed ports and persistent storage. This example uses a shared filesystem. Sensu Go is backed by a distributed database, and its storage should be provisioned accordingly. We recommend local storage or something like Throughput Optimized or Provisioned IOPS EBS if local storage is unavailable. The exposed ports are:

- ☐ 2380: Sensu storage peer listener (only other Sensu backends need access to this port)
- ☐ 3000: Sensu dashboard
- ☐ 8080: Sensu API (all users need access to this port)
- ☐ 8081: Agent API (all agents need access to this port)

We suggest, but do not require, persistent storage for Sensu backends and Sensu agents. The Sensu agent will cache runtime assets locally for each check, and the Sensu backend will cache runtime assets locally for each handler and mutator. This storage should be unique per sensu-backend/sensu-agent process.

Start a Sensu backend

```
docker run -v /var/lib/sensu:/var/lib/sensu -d --name sensu-backend -p 2380:2380
\
-p 3000:3000 -p 8080:8080 -p 8081:8081 sensu/sensu:latest sensu-backend start
```

Start a Sensu agent

In this case, we're starting an agent with the webserver and system subscriptions as an example.This assumes that the Sensu backend is running on another host named sensu.yourdomain.com.If you are running these locally on the same system, add `--link sensu-backend` to your Docker arguments, and change the backend URL to `--backend-url ws://sensu-backend:8081`.

```
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 \
--subscriptions webserver,system --cache-dir /var/lib/sensu
```

NOTE: You can configure the backend and agent log levels by using the `--log-level` flag on either process. Log levels include `panic`, `fatal`, `error`, `warn`, `info`, and `debug`, defaulting to `warn`.

sensuctl and Docker

It's best to run sensuctl locally and point it at the exposed API port for your the Sensu backend.The sensuctl utility stores configuration locally, and you'll likely want to persist it across uses.While it can be run from the docker container, doing so may be problematic.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Supported platforms

Sensu backend

The Sensu backend is available for 64-bit Linux.See the backend installation guide for more information.

Platform & Version	amd64
CentOS/RHEL 5	
CentOS/RHEL 6	
CentOS/RHEL 7	
Ubuntu 16.04	
Ubuntu 18.04	
Ubuntu 18.10	

Sensu agent

The Sensu agent is available for Linux and Windows.See the agent installation guide for more information.

Platform & Version	amd64

CentOS 5/RHEL
CentOS 6/RHEL
CentOS 7/RHEL
Ubuntu 16.04
Ubuntu 18.04
Ubuntu 18.10
Windows Server 2008 R2 and later

Sensuctl command-line tool

Sensuctl is available for Linux, Windows, and macOS.See the sensuctl installation guide for more information.

Platform & Version	`amd64`
CentOS 5/RHEL	
CentOS 6/RHEL	
CentOS 7/RHEL	
Ubuntu 16.04	
Ubuntu 18.04	
Ubuntu 18.10	
Windows 7 and later	
Windows Server 2008 R2 and later	

macOS 10.10 and later

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Installing Sensu Plugins

Sensu's functionality can be extended through the use of plugins.Plugins can provide executables for performing status or metric checks, mutators for changing data to a desired format, or handlers for performing an action on a Sensu event.

Installing plugins using Bonsai, the Sensu asset index

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins.You can use assets to provide the plugins, libraries, and runtimes you need to automate your monitoring workflows.Visit Bonsai to discover, download, and share assets, and get started with these helpful resources.

- Bonsai, the Sensu asset index
- Installing plugins with assets
- Sharing assets on Bonsai

Installing plugins using the sensu-install tool

You can find a number of plugins in the Sensu Plugins organization on GitHub.

NOTE: Plugins found in the Sensu Plugins GitHub organization are community-maintained, meaning that anyone can improve on a plugin found there. If you have a question about how you can get involved in adding to, or providing a plugin, head to the Sensu Community Slack channel. Maintainers are always happy to help answer questions and point you in the right direction.

To install a Sensu Community Plugin with Sensu Go:

1. Install the sensu-plugins-ruby package from packagecloud.

2. Use the `sensu-install` command to install any plugins in the Sensu Plugins organization on GitHub by repository name. Plugins are installed into `/opt/sensu-plugins-ruby/embedded/bin`.

```
sensu-install --help
Usage: sensu-install [options]
    -h, --help                       Display this message
    -v, --verbose                    Enable verbose logging
    -p, --plugin PLUGIN              Install a Sensu PLUGIN
    -P, --plugins PLUGIN[,PLUGIN]    PLUGIN or comma-delimited list of Sensu
plugins to install
    -e, --extension EXTENSION        Install a Sensu EXTENSION
    -E, --extensions EXTENSION[,EXT] EXTENSION or comma-delimited list of Sensu
extensions to install
    -s, --source SOURCE              Install Sensu plugins and extensions from a
custom SOURCE
    -c, --clean                      Clean up (remove) other installed versions
of the plugin(s) and/or extension(s)
    -x, --proxy PROXY                Install Sensu plugins and extensions via a
PROXY URL
```

For example, to install the Sensu Disk Checks Plugin:

```
sudo sensu-install -p disk-checks
```

To install a specific version of the Sensu Disk Checks Plugin with `sensu-install`, run:

```
sudo sensu-install -p 'sensu-plugins-disk-checks:3.1.0'
```

We strongly recommend using a configuration management tool or using Sensu assets to pin the versions of any plugins installed in production.

NOTE: Sensu Go is compatible with all check executables in the Sensu Plugins organization. Handler and mutator executables are not yet compatible with Sensu Go.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Hardware requirements

- Sensu backend requirements
- Sensu agent requirements
- Networking recommendations
- Cloud recommendations

Sensu backend

Backend minimum requirements

The following configuration is the minimum required to run the Sensu backend, however it is insufficient for production use.See the recommended configuration for production recommendations.

- 64-bit Intel or AMD CPU
- 4 GB RAM
- 4 GB free disk space
- 10 mbps network link

Backend recommended configuration

The following configuration is recommended as a baseline for production use to ensure a good user and operatorexperience. Using additionalresources (even over-provisioning) further improves stability andscalability.

- 64 bit 4-core Intel or AMD CPU

- 8 GB RAM
- SSD (NVMe or SATA3)
- Gigabit ethernet

The Sensu backend is typically CPU and storage intensive. In general, its use ofthese resources scales linearly with the total number ofchecks executed by all Sensu agents connecting to the backend.

The Sensu backend is a massively parallel application that can scale toany number of CPU cores. Provision roughly 1 CPU core for every 50checks per second (including agent keepalives).Most installations are fine with 4 CPU cores, but larger installationsmay find that additional CPU cores (8+) are necessary.

Every executed Sensu check results in storage writes. Whenprovisioning storage, a good guideline is to have twice as many**sustained disk IOPS** as you expect to have events per second. Don'tforget to include agent keepalives in this calculation; each agentpublishes a keepalive every 20 seconds. For example, in a cluster of 100 agents,you can expect those agents to consume 10 write IOPS for keepalives.

The Sensu backend uses a relatively modest amount of RAM under mostcircumstances. Larger production deployments use a larger amountof RAM (8+ GB).

Sensu agent

Agent minimum requirements

The following configuration is the minimum required to run the Sensu agent, however it is insufficient for production use.See the recommended configuration for production recommendations.

- 386, amd64, or ARM CPU (armv5 minimum)
- 128 MB RAM
- 10 mbps network link

Agent recommended configuration

The following configuration is recommended as a baseline for production use to ensure a good user and operator experience.

- 64 bit 4-core Intel or AMD CPU
- 512 MB RAM
- Gigabit ethernet

The Sensu agent itself is quite lightweight, and should be able to runon all but the most modest hardware. However, since the agent isresponsible for executing checks, factor the agent's responsibilitiesinto 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.

It's recommended that users connect backends and agents via gigabitethernet, but any somewhat-reliable network link should work (e.g.WiFi and 4G). If you see WebSocket timeouts in the backend logs, youmay need to use a better network link between the backend and agents.

Cloud recommendations

AWS

The recommended EC2 instance type and size for Sensu backends runningembedded etcd is **M5d.xlarge**. The M5d instance provides4 vCPU, 16 GB of RAM, up to 10 Gbps network connectivity, and a 150NVMe SSD directly attached to the instance host (optimal for sustaineddisk IOPS).

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Upgrading Sensu

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

To upgrade to the latest version of Sensu Go from version 5.0.0 or later, first install the latest packages.

Then restart the services.

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

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

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

You can use the `version` command to determine the installed version using the `sensu-agent`, `sensu-backend`, and `sensuctl` tools. For example: `sensu-backend version`.

Migrating to Sensu Go from Sensu Core 1.x

This guide provides general information for migrating your Sensu instance from Sensu Core 1.x to Sensu Go 5.0.For instructions and tools to help you translate your Sensu configuration from Sensu Core 1.x to Sensu Go, see the following resources.

- Sensu translator project
- Jef Spaleta - Check configuration upgrades with the Sensu Go sandbox

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

- Packaging
- Architecture
- Entities
- Checks
- Events
- Handlers
- Filters
- Assets
- Role-based access control
- Silencing
- Token substitution
- Aggregates
- API
- Custom attributes

Packaging

Sensu is now provided as three packages: sensu-go-backend, sensu-go-agent, and sensu-go-cli (sensuctl).This results in a fundamental change in Sensu terminology from Sensu Core 1.x: the server is now the backend; the client is now the agent.To learn more about new terminology in Sensu Go, see the glossary.

Architecture

The external RabbitMQ transport and Redis datastore in Sensu Core 1.x have been replaced with an embedded transport and etcd datastore in Sensu Go.The Sensu backend and agent are configured using YAML files or using the `sensu-backend` or `sensu-agent` command-line tools, instead of using JSON files.Sensu checks and pipeline elements are now configured via the API or sensuctl tool instead of JSON files.See the backend, agent, and sensuctl reference docs for more information.

Entities

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

Checks

Standalone checks are no longer supported in Sensu Go, although similar functionality can be achieved using role-based access control, assets, and entity subscriptions .There are also a few changes to check definitions to be aware of. The `stdin` check attribute is no longer supported in Sensu Go, and Sensu Go no longer tries to run a “default” handler when executing a check without a specified handler. Additionally, round-robin subscriptions and check subdues are not yet available in Sensu Go.

Check hooks are now a resource type in Sensu Go, meaning that hooks can be created, managed, and reused independently of check definitions.You can also execute multiple hooks for any given response code.

Events

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

Handlers

Transport handlers are no longer supported by Sensu Go, but you can create similar functionality

using a pipe handler that connects to a message bus and injects event data into a queue.

Filters

Ruby eval logic has been replaced with JavaScript expressions in Sensu Go, opening up powerful possibilities to filter events based on occurrences and other event attributes.As a result, the built-in occurrences filter in Sensu Core 1.x is not provided in Sensu Go, but you can replicate its functionality using this filter definition.Sensu Go includes three new built-in filters: only-incidents, only-metrics, and allow-silencing.Sensu Go does not yet include a built-in check dependencies filter or a filter-when feature.

Assets

The sensu-install tool has been replaced in Sensu Go by assets, shareable, reusable packages that make it easy to deploy Sensu plugins.Sensu Plugins in Ruby can still be installed via sensu-install by installing sensu-plugins-ruby; see the installing plugins guide for more information.

Role-based access control

Role-based access control (RBAC) is a built-in feature of the open-source version of Sensu Go.RBAC allows management and access of users and resources based on namespaces, groups, roles, and bindings.To learn more about setting up RBAC in Sensu Go, see the RBAC reference and the guide to creating a read-only user.

Silencing

Silencing is now disabled by default in Sensu Go and must be enabled explicitly using the built-in `not_silenced` filter.

Token substitution

The syntax for using token substitution has changed from using triple colons to using double curly braces.

Aggregates

Sensu Go does not yet support check aggregates.

API

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

Custom attributes

Custom check attributes are no longer supported in Sensu Go.Instead, Sensu Go provides the ability to add custom labels and annotations to entities, checks, assets, hooks, filters, mutators, handlers, and silencing entries.See the metadata attributes section in the reference documentation for more information about using labels and annotations (for example: metadata attributes for entities).

For Containers

For Cloud

For Nagios

Pricing

Verifying Sensu downloads

All Platforms

Sensu tar archives are available for Linux, Windows, and macOS.See the installation guide for more information.

You can verify a Sensu download using SHA-512 checksums.

Windows

Download Sensu for Windows.

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.0.1/sensu-go-5.0.1-windows-amd64.tar.gz  -OutFile "$env:userprofile\sensu-go-5.0.1-windows-amd64.tar.gz"
```

Generate a SHA-512 checksum for the downloaded artifact.

```
Get-FileHash "$env:userprofile\sensu-go-5.0.1-windows-amd64.tar.gz" -Algorithm SHA512 | Format-List
```

The result should match (with the exception of capitalization) the output from the following commands.

```
Invoke-WebRequest https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.0.1/sensu-go-5.0.1-windows-amd64.sha512sum -OutFile "$env:userprofile\sensu-
```

```
go-5.0.1-windows-amd64.sha512sum"

Get-Content "$env:userprofile\sensu-go-5.0.1-windows-amd64.sha512sum"
```

macOS

Download Sensu for macOS.

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.0.1/sensu-go-5.0.1-darwin-amd64.tar.gz
```

Generate a SHA-512 checksum for the downloaded artifact.

```
shasum -a 512 sensu-go-5.0.1-darwin-amd64.tar.gz
```

The result should match the output from the following command.

```
curl -LO https://s3-us-west-2.amazonaws.com/sensu.io/sensu-go/5.0.1/sensu-go-5.0.1-darwin-amd64.sha512sum && cat sensu-go-5.0.1-darwin-amd64.sha512sum
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Supported versions

We recommend updating Sensu frequently to stay in sync with the latest features and fixes.See the upgrade guide to upgrade to the latest version.

Sensu supports the latest versions of official distributions, including packages, binary-only distributions, and Docker images.To learn more about Sensu support and licensing, see the getting started guide.

version	release date	status	end of support date
5.8.0 (docs)		Pre-release	
5.7.0 (docs)	May 9, 2019	Supported	
5.6.0 (docs)	April 30, 2019	Supported	
5.5.1 (docs)	April 17, 2019	Supported	
5.5.0 (docs)	April 4, 2019	Supported	
5.4.0 (docs)	March 27, 2019	Supported	
5.3.0 (docs)	March 11, 2019	Supported	
5.2.1 (docs)	February 11, 2019	Not recommended	
5.2.0 (docs)	February 7, 2019	Not recommended	
5.1.1 (docs)	January 24, 2019	Not recommended	
5.1.0 (docs)	December 19, 2018	Not recommended	

5.0.1 (docs)	December 12, 2018	Not recommended	June 3, 2019
5.0.0 (docs)	December 5, 2018	Not recommended	June 3, 2019

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Guides

- Aggregate-Metrics-Statsd
- Clustering
- Create-Read-Only-User
- Enrich-Events-With-Hooks
- Extract-Metrics-With-Checks
- Influx-Db-Metric-Handler
- Install-Check-Executables-With-Assets
- Monitor-External-Resources
- Monitor-Server-Resources
- Plan-Maintenance
- Reduce-Alert-Fatigue
- Securing-Sensu
- Send-Slack-Alerts
- Systemd-Logs
- Troubleshooting

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

How to aggregate metrics with the Sensu StatsD listener

What is StatsD?

StatsD, originating from the daemon written by Etsy, is a daemon, tool,and protocol that can be used to send, collect, and aggregate custom metrics.Services that implement StatsD typically expose UDP port 8125 to receive metricsaccording to the line protocol `<metricname>:<value>|<type>`.

Why use StatsD?

StatsD allows you to measure anything and everything. You can monitorapplication performance by collecting custom metrics in your code and sendingthem to a StatsD server or you can monitor system levels of CPU, I/O, networketc. with collection daemons. The metrics that StatsD aggregates can be fed tomultiple different backends to store or visualize the data.

How does Sensu implement StatsD?

Sensu implements a StatsD listener on its agents. Each `sensu-agent` listens on the default port 8125 for UDP messages which follow the StatsD lineprotocol. StatsD aggregates the metrics, and Sensu translates them to Sensumetrics and events to be passed to the event pipeline. The listener is configurable (see Configuring the StatsD listener)and can be accessed 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, soit is important to configure an event handler(s).

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

Configuring the StatsD listener

The Sensu StatsD Server is configured at the start-up of a `sensu-agent`. Theflags below allow you to configure the 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 the followingresources to learn how to handle those metrics:

- Read the handlers reference for in-depth documentation on handlers.
- Read the InfluxDB handler guide for instructions on Sensu's built-inmetric handler.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

How to run a Sensu cluster

What is a Sensu cluster?

A Sensu cluster is a group of at least three sensu-backend nodes, each connected to a shared etcd cluster, using Sensu's embedded etcd or an external etcd cluster. Creating a Sensu cluster ultimately configures an etcd cluster.

Why use clustering?

Clustering is important to make Sensu more highly available, reliable, and durable. It will help you cope with the loss of a backend node, prevent data loss, and distribute the network load of agents.

NOTE: We recommend using a load balancer to evenly distribute agent connections across the cluster.

Configuring a cluster

The sensu-backend arguments for its store mirror the etcd configuration flags, however the Sensu flags are prefixed with `etcd`. For more detailed descriptions of the different arguments, you can refer to the etcd docs or the Sensu backend reference.

You can configure a Sensu cluster in a couple different ways (we'll show you a few below) but it's recommended to 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 (Core OS).

We also recommend using stable platforms to support your etcd instances (see etcd's supported platforms).

Docker

If you'd 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 uses on disk configuration. If you are using an ephemeral computer instance, you can use `sensu-backend start --help` to see examples of etcd command line flags. The configuration file entries below translate to `sensu-backend` flags.

Sensu backend configuration

Below are example 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. You can use our installation and configuration guide guide if you have not done so.

backend-1

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

backend-2

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

```
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: ""
etcd-name: "backend-2"
```

backend-3

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

Once each node has the configuration described above, start each sensu-backend:

```
sudo systemctl start sensu-backend
```

Adding sensu agents to the cluster

Each Sensu agent should have the following entries in `/etc/sensu/agent.yml` to ensure they are aware of all cluster members. This allows the agent to reconnect to a working backend in the scenrio where the one it is currently connected to goes into an unhealthy state.

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

backend-url:
```

```
  - "ws://10.0.0.1:8081"
  - "ws://10.0.0.2:8081"
  - "ws://10.0.0.3:8081"
```

You should now have a highly available Sensu cluster! You can verify its health and try other cluster management commands using sensuctl.

Sensuctl

Sensuctl has several commands to help you manage and monitor your cluster. See `sensuctl cluster -h` for additional help usage.

Cluster health

Get cluster health status and etcd alarm information.

```
sensuctl cluster health

        ID            Name                    Error                       Healthy
 ────────────────── ────────── ─────────────────────────────────────────── ─────────
  a32e8f613b529ad4   backend-1                                             true
  c3d9f4b8d0dd1ac9   backend-2  dial tcp 10.0.0.2:2379: connect: connection refused   false
  c8f63ae435a5e6bf   backend-3                                             true
```

Add a cluster member

Add a new member node to an existing cluster.

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

added member 2f7ae42c315f8c2d to cluster
```

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

List cluster members

List the ID, name, peer urls, and client urls of all nodes in a cluster.

```
sensuctl cluster member-list

       ID            Name          Peer URLs               Client URLs
 ────────────────── ─────────── ───────────────────── ─────────────────────
  a32e8f613b529ad4   backend-1   https://10.0.0.1:2380   https://10.0.0.1:2379
  c3d9f4b8d0dd1ac9   backend-2   https://10.0.0.2:2380   https://10.0.0.2:2379
  c8f63ae435a5e6bf   backend-3   https://10.0.0.3:2380   https://10.0.0.3:2379
  2f7ae42c315f8c2d   backend-4   https://10.0.0.4:2380   https://10.0.0.4:2379
```

Remove a cluster member

Remove a faulty or decommissioned member node from a cluster.

```
sensuctl cluster member-remove 2f7ae42c315f8c2d

Removed member 2f7ae42c315f8c2d from cluster
```

Update a cluster member

Update the peer URLs of a member in a cluster.

```
sensuctl cluster member-update c8f63ae435a5e6bf https://10.0.0.4:2380
```

```
Updated member with ID c8f63ae435a5e6bf in cluster
```

Security

Creating self-signed certificates

We will use the cfssl tool to generate our self-signed certificates.

The first step is to create a **Certificate Authority (CA)**. In order to keep things simple, we will generate all our clients and peer certificates using this CA, but you might eventually want to create distinct CA.

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

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

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

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

export ADDRESS=10.0.0.3,backend-3
```

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

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

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

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

Once done, you should have the following files created. The `*.csr` files will not be used in this guide.

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

Client-to-server transport security with HTTPS

Below are example configuration snippets from `/etc/sensu/backend.yml` on three Sensu backends 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.This configuration assumes that your client certificates are in `/etc/sensu/certs/` and your CA certificate is in `/usr/local/share/ca-certificates/sensu/`.

```
##
# etcd peer ssl configuration for backend-1/10.0.0.1
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-1.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-1-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"

##
# etcd peer ssl configuration for backend-2/10.0.0.2
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-2.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-2-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"

##
# etcd peer ssl configuration for backend-3/10.0.0.3
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-3.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-3-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
```

Validating with curl:

```
curl --cacert /usr/local/share/ca-certificates/sensu/ca.pem \
https://127.0.0.1:2379/v2/keys/foo -XPUT -d value=bar
```

Client-to-server authentication with HTTPS client certificates

Below are example configuration snippets from `/etc/sensu/backend.yml` on three Sensu backends 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.This configuration assumes your client certificates are in `/etc/sensu/certs/` and your CA certificate is in `/usr/local/share/ca-certificates/sensu/`.

```
##
# etcd peer ssl configuration for backend-1/10.0.0.1
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-1.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-1-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
etcd-client-cert-auth: true

##
# etcd peer ssl configuration for backend-2/10.0.0.2
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-2.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-2-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
etcd-client-cert-auth: true

##
# etcd peer ssl configuration for backend-3/10.0.0.3
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-3.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-3-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
etcd-client-cert-auth: true
```

Validating with curl, with a different certificate and key:

```
curl --cacert /usr/local/share/ca-certificates/sensu/ca.pem \
--cert /etc/sensu/certs/client.pem \
```

```
--key /etc/sensu/certs/client-key.pem \
-L https://127.0.0.1:2379/v2/keys/foo -XPUT -d value=bar
```

Peer communication authentication with HTTPS client certificates

Below are example configuration snippets from `/etc/sensu/backend.yml` on three Sensu backends 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: If you ran through the first part of the guide, you will need to update the store configuration for all backends to use http **s** *instead of http.*

backend-1

```
##
# store configuration for backend-1/10.0.0.1
##

etcd-listen-client-urls: "https://10.0.0.1:2379"
etcd-listen-peer-urls: "https://0.0.0.0:2380"
etcd-initial-cluster: "backend-1=https://10.0.0.1:2380,backend-
2=https://10.0.0.2:2380,backend-3=https://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "https://10.0.0.1:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: "sensu"
etcd-name: "backend-1"

##
# etcd peer ssl configuration for backend-1/10.0.0.1
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-1.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-1-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
etcd-peer-client-cert-auth: true
```

backend-2

```
##
# store configuration for backend-2/10.0.0.2
##

etcd-listen-client-urls: "https://10.0.0.2:2379"
etcd-listen-peer-urls: "https://0.0.0.0:2380"
etcd-initial-cluster: "backend-1=https://10.0.0.1:2380,backend-
2=https://10.0.0.2:2380,backend-3=https://10.0.0.3:2380"
etcd-initial-advertise-peer-urls: "https://10.0.0.2:2380"
etcd-initial-cluster-state: "new"
etcd-initial-cluster-token: "sensu"
etcd-name: "backend-2"

##
# etcd peer ssl configuration for backend-2/10.0.0.2
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-2.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-2-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
etcd-peer-client-cert-auth: true
```

backend-3

```
##
# store configuration for backend-3/10.0.0.3
##

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

```
##
# etcd peer ssl configuration for backend-3/10.0.0.3
##

etcd-peer-cert-file: "/etc/sensu/certs/backend-3.pem"
etcd-peer-key-file: "/etc/sensu/certs/backend-3-key.pem"
etcd-peer-trusted-ca-file: "/usr/local/share/ca-certificates/sensu/ca.pem"
etcd-peer-client-cert-auth: true
```

Sensu agent with HTTPS

Below is a sample configuration for an agent that would connect to the cluster using `wss` from `/etc/sensu/agent.yml`.

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

backend-url:
  - "wss://10.0.0.1:8081"
  - "wss://10.0.0.2:8081"
  - "wss://10.0.0.3:8081"
```

Using an external etcd cluster

To stand up an external etcd cluster, you can follow etcd's clustering guide using the same store configuration.

In this example, we will enable client-to-server and peer communication authentication using self-signed TLS certificates. Below is how you would start etcd for `backend-1` from our three node configuration example above.

```
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
```

In order to inform Sensu that you'd like to use this external etcd data source, add the `sensu-backend` flag `--no-embed-etcd` to the original configuration, along with the path to a client certificate created using our CA.

```
sensu-backend start \
--etcd-trusted-ca-file=./ca.pem \
--etcd-cert-file=./client.pem \
--etcd-key-file=./client-key.pem \
--etcd-advertise-client-
urls=https://10.0.0.1:2379,https://10.0.0.2:2379,https://10.0.0.3:2379 \
--no-embed-etcd
```

Troubleshooting

Failures modes

See the etcd failure modes documentation for more information.

Disaster recovery

See the etcd recovery guide for more information.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to create a read-only user with RBAC

Sensu role-based access control (RBAC) helps different teams and projects share a Sensu instance. RBAC allows management and access of users and 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 the default `admin` user.

Why use RBAC?

RBAC allows you to exercise fine-grained control over how Sensu users interactwith Sensu resources. Using RBAC rules, you can easily achieve **multitenancy**so different projects and teams can share a Sensu instance.

How to create a read-only user

In this section, you'll create a user and assign them read-only access to resources within the `default` namespace using a **role** and a **role binding**.

1. Create a user with the username `alice` and assign them to the group `ops`:

```
sensuctl user create alice --password='password' --groups=ops
```

2. Create a `read-only` role with `get` and `list` permissions for all resources (`*`) within the `default` namespace:

```
sensuctl role create read-only --verb=get,list --resource=* --
namespace=default
```

3. Create an `ops-read-only` role binding to assign the `read-only` role to the `ops` group:

```
sensuctl role-binding create ops-read-only --role=read-only --group=ops
```

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

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

How to create a cluster-wide event-reader user

Now let's say you want to create a user that has read-only access to events across all namespaces. Since you want this role to have cluster-wide permissions, you'll need to create a **cluster role** and a **cluster role binding**.

1. Create a user with the username `bob` and assign them to the group `ops`:

```
sensuctl user create bob --password='password' --groups=ops
```

2. Create a `global-event-reader` cluster role with `get` and `list` permissions for `events` across all namespaces:

```
sensuctl cluster-role create global-event-reader --verb=get,list --
resource=events
```

3. Create an `ops-event-reader` cluster role binding to assign the `global-event-reader` role to the `ops` group:

```
sensuctl cluster-role-binding create ops-event-reader --cluster-
role=global-event-reader --group=ops
```

All users in the `ops` group now have read-only access to events across all namespaces.

Next steps

You now know how to create a user, create a role, and create a role binding to assign a role to a user. From this point, here are some recommended resources:

- Read the RBAC reference for in-depth documentation on role-based access control, examples, and information about cluster-wide permissions.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to augment event data using check hooks

What are check hooks?

Check hooks are **commands** run by the Sensu agent in response to the result of **check** command execution. The Sensu agent executes the appropriateconfigured hook, depending on the exit status code (e.g., `1`).

Why use check hooks?

Check hooks allow Sensu users to automate data collection routinely performed byoperators investigating monitoring alerts, freeing precious operator time! Whilecheck hooks can be used for rudimentary auto-remediation tasks, they are intendedfor enrichment of monitoring event data.

Using check hooks to gather context

The purpose of this guide is to help you put in place a check hook which capturesthe process tree in the event that an `nginx_process` check returns a status of `2` (critical,not running).

Creating the hook

The first step is to create a new hook that runs a specific command tocapture the process tree. We can set an execution **timeout** of 10 secondsfor this command.

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

Assigning the hook to a check

Now that the `process_tree` hook has been created, it can be assigned to acheck. Here we apply our hook to an already existing `nginx_process` check.By setting the `type` to `critical`, we ensure 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
```

Validating the check hook

You can verify the proper behavior of the check hook against a specific event byusing `sensuctl`. It might take a few moments, once the check hook is assigned,for the check to be scheduled on the entity and the result sent back to the Sensubackend. The check hook command result is available in the `hooks` array,within the `check` scope.

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

{
  [...]
  "check": {
    [...]
    "hooks": [
      {
        "config": {
          "name": "process_tree",
          "command": "ps aux",
          "timeout": 10,
          "namespace": "default"
```

```
        },
        "duration": 0.008713605,
        "executed": 1521724622,
        "output": "",
        "status": 0
      }
    ],
    [...]
  }
}
```

Having confirmed that the hook is attached to our check, we can stopNginx and observe the check hook in action on the next checkexecution. Here we use sensuctl to query event info and send the response to `jq` so we can isolate the check hook output:

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

USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3  46164  6704 ?        Ss   Nov17   0:11
/usr/lib/systemd/systemd --switched-root --system --deserialize 20
root          2  0.0  0.0      0     0 ?        S    Nov17   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    Nov17   0:01 [ksoftirqd/0]
root          7  0.0  0.0      0     0 ?        S    Nov17   0:01 [migration/0]
root          8  0.0  0.0      0     0 ?        S    Nov17   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        S    Nov17   0:34 [rcu_sched]
```

Note that the above output, although truncated in the interest ofbrevity, reflects the output of the `ps aux` command specified in thecheck hook we created. Now when we are alerted that Nginx is notrunning, we can review the check hook output to confirm this was thecase, without ever firing up an SSH session to investigate!

Next steps

You now know how to run data collection tasks using check hooks. From this point,here are some recommended resources:

- Read the hooks reference for in-depth documentation on hooks.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to collect and extract metrics using Sensu checks

What are Sensu checks?

In short, Sensu checks are **commands** (or scripts), executed by the Sensuagent, that output data and produce an exit code to indicate a state. If you areunfamiliar with checks, or would like to learn how to configure one first,take a look through the check reference doc and guide before youcontinue.

Extracting metrics from check output

In order to extract metrics from check output, you'll need to do the following:

1. Configure the check `command` such that the command execution outputsmetrics in one of the supported output metric formats.
2. Configure the check `output_metric_format` to one of the supported output metric formats.
3. Configure the check `output_metric_handlers` (optional) to a Sensu handlerthat is equipped to handle Sensu metrics (see handlers or influx-db handler to learn more).

You can configure the check with these fields at creation, or use the commandsbelow assuming you have a check named `collect-metrics`. In this example,we'll be using `graphite_plaintext` format and sending the metrics to a handlernamed `influx-db`.

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

Supported output metric formats

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

nagios

output_metric_format	`nagios_perfdata`
documentation	Nagios Performance Data
example	`PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80`

graphite

output_metric_format	`graphite_plaintext`
documentation	Graphite Plaintext Protocol
example	`local.random.diceroll 4 123456789`

influxdb

output_metric_format	`influxdb_line`
documentation	InfluxDB Line Protocol
example	`weather,location=us-midwest temperature=82 1465839830100400200`

opentsdb	
output_metric_format	`opentsdb_line`
documentation	OpenTSDB Data Specification
example	`sys.cpu.user 1356998400 42.5 host=webserver01 cpu=0`

Validating the metrics

If the check output is formatted correctly according to its `output_metric_format`,the metrics will be extracted in Sensu Metric Format, and saved within theevent. You should expect to see logged errors if Sensu is unable to parsethe check output. You can validate that metrics have been extracted from yourcheck through your handler, or through the resulting event. The example checkwe used would yield an event similar to the one below:


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

```
    "metrics": {
      "handlers": [
        "influx-db"
      ],
      "points": [
        {
          "name": "cpu.idle_percentage",
          "value": 61,
          "timestamp": 1525462242,
          "tags": []
        },
        {
          "name": "mem.sys",
          "value": 104448,
          "timestamp": 1525462242,
          "tags": []
        }
      ]
    }
  }
}
```

Next steps

Now you know how to extract metrics from check output! Check out the belowresources for some further reading:

- Read the checks reference for in-depth documentation on checks.
- Read the checks guide for directions on how to schedule checks.
- Read the handlers reference for in-depth documentation on handlers.
- Read the influx-db handler guide for instructions on Sensu's built-inmetric handler.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

How to populate InfluxDB metrics using handlers

What are Sensu handlers?

Sensu event handlers are actions executed by the Sensu server on events.In this example, we'll use a handler to populate a time series database. Ifyou're not totally comfortable with handlers yet, check out the in-depthguide on handlers first!

Using a handler to populate InfluxDB

The purpose of this guide is to help you populate Sensu metrics into the timeseries database InfluxDB. Metrics can be collected from check outputor from the Sensu StatsD Server.

Registering the asset

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins.In this guide, we'll use the Sensu InfluxDB handler asset to power an `influx-db` handler.

You can use the following sensuctl example to register the Sensu InfluxDB handler asset for Linux AMD64, or you can download the latest asset definition for your platform from Bonsai and register the asset using `sensuctl create --file filename.yml`.

```
sensuctl asset create sensu-influxdb-handler --url "https://github.com/sensu/sensu-i
influxdb-handler_3.1.2_linux_amd64.tar.gz" --sha512
"612c6ff9928841090c4d23bf20aaf7558e4eed8977a848cf9e2899bb13a13e7540bac2b63e324f39d9
```

You should see a confirmation message from sensuctl.

```
Created
```

Creating the handler

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

```
sensuctl handler create influx-db \
--type pipe \
--command "sensu-influxdb-handler -d sensu" \
--env-vars "INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086,
INFLUXDB_USER=sensu, INFLUXDB_PASS=password" \
--runtime-assets sensu-influxdb-handler
```

You should see a confirmation message from sensuctl.

```
Created
```

Assigning the handler to an event

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

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

The handler can also be assigned to the Sensu StatsD listener at agent startup to passall StatsD metrics into InfluxDB:

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

Validating the handler

It might take a few moments once the handler is assigned to the check or StatsDserver, for Sensu to receive the metrics, but once an event is handled, youshould start to see your InfluxDB being populated! Otherwise, you can verify theproper behavior of this handler by using `sensu-backend` logs. See the troubleshooting guide 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 one with the message `"msg":"pipelined executed event pipeHandler","output":"","status":0` .

Next steps

You now know how to apply a handler to metrics and take action on events. Fromthis point, here are some recommended resources:

- Read the handlers reference for in-depth documentation on handlers.
- Read the StatsD listener guide for instructions on how to aggregateStatsD metrics in Sensu.
- Read the check output metric extraction guide to learn how to collectand extract metrics using Sensu checks.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

How to install plugins using assets

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins.You can use assets to provide the plugins, libraries, and runtimes you need to power your monitoring workflows. See the asset reference for more information about assets.

1. Download an asset definition from Bonsai

You can discover, download, and share assets using Bonsai, the Sensu asset index.To use an asset, select the Download button on the asset page in Bonsai to download the asset definition for your Sensu backend platform and architecture.Asset definitions tell Sensu how to download and verify the asset when required by a check, filter, mutator, or handler.

For example, here's the asset definition for version 1.1.0 of the Sensu PagerDuty handler asset for Linux AMD64.

```
---
type: Asset
api_version: core/v2
metadata:
  name: sensu-pagerduty-handler
  namespace: default
  labels: {}
  annotations: {}
spec:
```

```
  url: https://github.com/sensu/sensu-pagerduty-handler/releases/download/1.1.0/sen
  sha512:
e93ec4465af5a2057664e8c3cd68e9352457b81315b97578eaae5e21f0cf7419d4fc36feb0155eeb0dd
  filters:
  - entity.system.os == 'linux'
  - entity.system.arch == 'amd64'
```

Enterprise-only assets (like the ServiceNow and Jira event handlers) require an active enterprise license. For more information about enterprise-only features and to active your license, see the getting started guide.

2. Register the asset with Sensu

Once you've downloaded the asset definition, you can register the asset with Sensu using sensuctl.

```
sensuctl create --file sensu-sensu-pagerduty-handler-1.1.0-linux-amd64.yml
```

You can use sensuctl to verify that the asset is registered and ready to use.

```
sensuctl asset list
```

3. Create a workflow

Now we can use assets in a monitoring workflow.Depending on the asset, you may want to create Sensu checks, filters, mutators, and handlers.The asset details in Bonsai are the best resource for information about asset capabilities and configuration.

For example, to use the Sensu PagerDuty handler asset, create a `pagerduty` handler that includes your PagerDuty service API key in place of `SECRET` and `sensu-pagerduty-handler` as a runtime asset.

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

```
    "type": "Handler",
    "metadata": {
        "namespace": "default",
        "name": "pagerduty"
    },
    "spec": {
        "type": "pipe",
        "env_vars": [
          "PAGERDUTY_TOKEN=SECRET"
        ],
        "runtime_assets": ["sensu-pagerduty-handler"],
        "timeout": 10,
        "filters": [
            "is_incident"
        ]
    }
}
```

Save the definition to a file (for example: `pagerduty-handler.json`), and add to Sensu using sensuctl.

```
sensuctl create --file pagerduty-handler.json
```

Now that Sensu can create incidents in PagerDuty, we can automate this workflow by adding the `pagerduty` handler to our Sensu service checks.To get started with checks, see the guide to monitoring server resources.

Next steps

- Learn more about assets
- Read the asset specification
- Share your assets on Bonsai

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to monitor external resources with proxy entities

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

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 read and create permissions for entities, checks, and events.

Using a proxy entity to monitor a website

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

Installing an HTTP check script

First, we'll install a bash script, named `http_check.sh`, to perform an HTTPcheck using **curl**.

```
sudo curl https://raw.githubusercontent.com/sensu/sensu-
go/5.1.0/examples/checks/http_check.sh \
```

```
-o /usr/local/bin/http_check.sh && \
sudo chmod +x /usr/local/bin/http_check.sh
```

PRO TIP: While this command may be appropriate when running a few agents, you should considerusing Sensu assets or a configuration management tool to provideruntime dependencies.

Creating the check

Now that our script is installed, we'll create a check named `check-http`, which runs the command `http_check.sh https://sensu.io`, at aninterval of 60 seconds, for all entities subscribed to the `proxy` subscription, using the `sensu-site` proxy entity name.

Create a file called `check.json` and add the following check definition.

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-http",
    "namespace": "default"
  },
  "spec": {
    "command": "http_check.sh https://sensu.io",
    "interval": 60,
    "proxy_entity_name": "sensu-site",
    "publish": true,
    "subscriptions": [
      "proxy"
    ]
  }
}
```

Now we can use sensuctl to add this check to Sensu.

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

```
sensuctl check list
    Name              Command              Interval   Cron   Timeout   TTL   Subscriptions   Handlers   Assets   Hooks
Publish?   Stdin?   Metric Format   Metric Handlers
────────────── ───────────────────────────────────── ──────────── ─────── ────────── ──────
───────────────── ──────────── ───────── ──────── ──────────── ───────── ─────────────────
───────────────────
 check-http   http_check.sh https://sensu.io       60              0    0   proxy                                true      false
```

Adding the subscription

To run the check, we'll need a Sensu agent with the subscription `proxy`. After [installing an agent](), open `/etc/sensu/agent.yml` and add the `proxy` subscription so the subscription configuration looks like:

```
subscriptions:
  - "proxy"
```

Then restart the agent.

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

Validating the check

Now we can use sensuctl to see that Sensu has created the proxy entity `sensu-site`.

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

And that Sensu is now monitoring `sensu-site` using the `check-http` check.

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

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

We can also see our new proxy entity in the Sensu dashboard.

Using proxy requests to monitor a group of websites

Now let's say that, instead of monitoring just sensu.io, we want to monitor multiple sites, for example: docs.sensu.io, packagecloud.io, and github.com.In this section of the guide, we'll use the `proxy_requests` check attribute, along with entity labels and token substitution, to monitor three sites using the same check.Before we get started, go ahead and install the HTTP check script if you haven't already.

Installing an HTTP check script

If you haven't already, install a bash script, named `http_check.sh`, to perform an HTTPcheck using **curl**.

```
sudo curl https://raw.githubusercontent.com/sensu/sensu-
go/5.1.0/examples/checks/http_check.sh \
-o /usr/local/bin/http_check.sh && \
sudo chmod +x /usr/local/bin/http_check.sh
```

PRO TIP: While this command may be appropriate when running a few agents, you should considerusing Sensu assets or a configuration management tool to provideruntime dependencies.

Creating proxy entities

Instead of creating a proxy entity using the `proxy_entity_name` check attribute, we'll be using sensuctl to create proxy entities to represent the three sites we want to monitor.Our proxy entities need the `entity_class` attribute set to `proxy` to mark them as proxy entities as well as a few custom `labels` that we'll use to identify them as a group and pass in individual URLs.

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

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

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

PRO TIP: When creating proxy entities, you can add whatever custom labels make sense for your environment. For example, when monitoring a group of routers, you may want to add `ip_address` labels.

Now we can use sensuctl to add these proxy entities to Sensu.

```
sensuctl create --file entities.json

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

Creating a reusable HTTP check

Now that we have our three proxy entities set up, each with a `proxy_type` and `url` label, we can use proxy requests and token substitution to create a single check that monitors all three sites.

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

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-http-proxy-requests",
    "namespace": "default"
  },
  "spec": {
    "command": "http_check.sh {{ .labels.url }}",
    "interval": 60,
    "subscriptions": [
      "proxy"
    ],
    "publish": true,
    "proxy_requests": {
      "entity_attributes": [
        "entity.entity_class == 'proxy'",
        "entity.labels.proxy_type == 'website'"
      ],
      "splay": true,
      "splay_coverage": 90
    }
  }
}
```

Our `check-http-proxy-requests` check uses the `proxy_requests` attribute to specify the applicable entities.In our case, we want to run the `check-http-proxy-requests` check on all entities of entity class `proxy` and proxy type `website`.To make sure that Sensu runs the check for all applicable entities, we need to set the splay attribute to `true` with a splay coverage percentage value of `90`.This gives Sensu 90% of the check `interval`, 60 seconds in this case, to execute the check for all applicable entities.Since we're using this check to monitor multiple sites, we can use token substitution to apply the correct `url` in the check `command`.

Now we can use sensuctl to add this check to Sensu.

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

sensuctl check list
        Name                    Command             Interval   Cron   Timeout   TTL   Subscriptions   Handlers   Assets
Hooks   Publish?   Stdin?   Metric Format   Metric Handlers
────────────────────────────── ───────────────────────────────────── ─────────── ──────
────────── ───── ───────────────── ─────────── ───────── ──────── ─────────── ─────────
───────────────── ───────────────────
  check-http                http_check.sh https://sensu.io          60             0     0   proxy                          true
false
  check-http-proxy-requests   http_check.sh {{ .labels.url }}          60             0     0   proxy
true      false                                      true      false
```

Validating the check

Before validating the check, make sure that you've added the `proxy` subscription to a Sensu agent if you haven't already.

Now we can use sensuctl to see that Sensu is monitoring docs.sensu.io, packagecloud.io, and github.com using the `check-http-proxy-requests`.

```
sensuctl event list
     Entity               Check            Output   Status   Silenced          Timestamp
───────────────────── ────────────────────────────── ───────── ───────── ───────────
───────────────────────────────────
github-site        check-http-proxy-requests             0   false      2019-01-17 17:10:31 +0000 UTC
packagecloud-site   check-http-proxy-requests             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-proxy-requests             0   false      2019-01-17 17:06:59 +0000 UTC
```

Next steps

You now know how to run a proxy check to verify the status of a website, aswell as using proxy requests to run a check on two different proxy entities based on label evaluation.From this point, here are some recommended resources:

- Read the proxy checks reference for in-depth documentation on proxy checks.
- Read the guide to providing runtime dependencies to checks with assets.
- Read the guide to sending alerts to Slack with handlers.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to monitor server resources with checks

What are Sensu checks?

Sensu checks are **commands** (or scripts), executed by the Sensu agent, thatoutput data and produce an exit code to indicate a state. Sensu checks use thesame specification as **Nagios**, therefore, Nagios **check plugins** may beused with Sensu.

Why use a check?

You can use checks to monitor server resources, services, and applicationhealth (for example: is Nginx running?) as well as collect and analyze metrics (for example: how much disk space do I have left?).

Using checks to monitor a service

The purpose of this guide is to help you monitor server resources, morespecifically the CPU usage, by configuring a check named `check-cpu` with a **subscription** named `system`, in order to target all **entities** subscribedto the `system` subscription.This guide requires a Sensu backend and at least one Sensu agent running on Linux.

Registering assets

To power the check, we'll use the Sensu CPU checks asset and the Sensu Ruby runtime asset.

Use the following sensuctl example to register the `sensu-plugins-cpu-checks` asset for CentOS, or download the asset definition for Debian or Alpine from Bonsai and register the asset using `sensuctl create --file filename.yml`.

```
sensuctl asset create sensu-plugins-cpu-checks --url "https://github.com/sensu-plug
checks/releases/download/4.0.0/sensu-plugins-cpu-checks_4.0.0_centos_linux_amd64.ta
"518e7c17cf670393045bff4af318e1d35955bfde166e9ceec2b469109252f79043ed133241c4dc9650
```

Then use the following sensuctl example to register the `sensu-ruby-runtime` asset for CentOS, or download the asset definition for Debian or Alpine from Bonsai and register the asset using `sensuctl create --file filename.yml`.

```
sensuctl asset create sensu-ruby-runtime --url "https://github.com/sensu/sensu-ruby
runtime_0.0.5_centos_linux_amd64.tar.gz" --sha512
"1c9f0aff8f7f7dfcf07eb75f48c3b7ad6709f2bd68f2287b4bd07979e6fe12c2ab69d1ecf5d4b9b9ed
```

You can use sensuctl to confirm that both the `sensu-plugins-cpu-checks` and `sensu-ruby-runtime` assets are ready to use.

```
sensuctl asset list
        Name                          URL                          Hash
──────────────────────────
──────────────────────────────────────────────────────────────────────
──────────
 sensu-plugins-cpu-checks  //github.com/.../sensu-plugins-cpu-checks_4.0.0_centos_linux_amd64.tar.gz  518e7c1
 sensu-ruby-runtime      //github.com/.../sensu-ruby-runtime_0.0.5_centos_linux_amd64.tar.gz      1c9f0af
```

Creating the check

Now that the assets are registered, we'll create a check named `check-cpu`, which runs the command `check-cpu.rb -w 75 -c 90` using the `sensu-plugins-cpu-checks` and `sensu-ruby-runtime` assets, at an **interval** of 60 seconds, for all entities subscribed to the `system` subscription.This checks generates a warning event (`-w`) when CPU usage reaches 75% and a critical alert (`-c`) at 90%.

```
sensuctl check create check-cpu \
--command 'check-cpu.rb -w 75 -c 90' \
--interval 60 \
--subscriptions system \
--runtime-assets sensu-plugins-cpu-checks,sensu-ruby-runtime
```

Configuring the subscription

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

```
subscriptions:
  - system
```

Then restart the agent.

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

Validating the check

We can use sensuctl to see that Sensu is monitoring CPU usage using the `check-cpu`, returning an OK status (`0`).It might take a few moments, once the check is created,for the check to be scheduled on the entity and the event returned to Sensu backend.

```
sensuctl event list
    Entity      Check                                   Output                                       Status   Silen
────────────── ──────────

──────────── ─────────── ──────────────────────────────
 sensu-centos   check-cpu   CheckCPU TOTAL OK:  total=0.2 user=0.0 nice=0.0 system=0.2 idle=99.8 iowait=0.0 irq=0.0 s
16:42:28 +0000 UTC
```

Next steps

You now know how to run a simple check to monitor CPU usage. From this point,here are some recommended resources:

- Read the checks reference for in-depth documentation on checks.
- Read our guide on providing runtime dependencies to checks with assets.
- Read our guide on monitoring external resources with proxy checks and entities.
- Read our guide on sending alerts to Slack with handlers.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to plan maintenance windows using silencing

What is Sensu silencing?

As **check results** are processed by a Sensu server, the server executes eventhandlers to send alerts to personnel or otherwise relay **event data** toexternal services. Sensu's built-in **silencing**, along with the built-in `not_silenced` filter, provides the means to suppress execution of eventhandlers on an ad hoc basis.

When to use silencing

Silencing is used to prevent handlers configured with the `not_silenced` filterfrom being triggered based on the check name present in a check result or thesubscriptions associated with the entity that published the check result. Thiscan be desirable in many scenarios, giving operators the ability to quietincoming alerts while coordinating their response.

Sensu silencing entries make it possible to:

- Silence all checks on a specific entity
- Silence a specific check on a specific entity
- Silence all checks on entities with a specific subscription
- Silence a specific check on entities with a specific subscription
- Silence a specific check on every entity

Using silencing to plan maintenance

The purpose of this guide is to help you plan a maintenance window, by creatinga silenced entry for a specific entity named `i-424242` and its check named `check-http`, in order to prevent alerts as you restart or redeploy theservices associated with this entity.

Creating the silenced entry

The first step is to create a silenced entry that will silence the check `check-http` on an entity named `i-424242`, for a planned maintenance windowthat starts at **01:00**, on **Sunday**, and ends **1 hour** later. Yourusername will automatically be added as the **creator** of the silenced entry.

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

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

Validating the silenced entry

You can verify that the silenced entry against our entity, here named `i-424242`, has been properly created, by using `sensuctl`.

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

Once the silenced entry starts to take effect, events that are silenced will bemarked as so in `sensuctl events`.

```
sensuctl event list

  Entity      Check      Output     Status    Silenced       Timestamp
 ────────────── ───────── ───────── ──────────── ──────────
 ──────────────────────────────
```

```
i-424242    check-http    0    true    2018-03-16 13:22:16 -0400 EDT
```

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

Next steps

You now know how to create silenced entries to plan a maintenance and hopefullyavoid false positive. From this point, here are some recommended resources:

- Read the silencing reference for in-depth documentation on silenced entries.

For Nagios

Pricing

How to reduce alert fatigue with filters

What are Sensu filters?

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

Why use a filter?

Filters are commonly used to filter recurring events (i.e. to eliminatenotification noise) and to filter events from systems in pre-productionenvironments.

Using filters to reduce alert fatigue

The purpose of this guide is to help you reduce alert fatigue by configuring afilter named `hourly`, for a handler named `slack`, in order to prevent alertsfrom being sent to Slack every minute. If you don't already have a handler inplace, learn how to send alerts with handlers.

Creating the filter

The first step is to create a filter that we will call `hourly`, which matchesnew events (where the event's `occurrences` is equal to `1`) or hourly events(so every hour after the first occurrence, calculated with the check's `interval` and the event's `occurrences`).

Events in Sensu Go are handled regardless ofcheck execution status; even successful check events are passed through thepipeline. Therefore, it's necessary 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"
```

Assigning the filter to a handler

Now that the `hourly` filter has been created, it can be assigned to a handler.Here, since we want to reduce the number of Slack messages sent by Sensu, we will applyour filter to an already existing handler named `slack`, in addition to thebuilt-in `is_incident` filter so only failing events are handled.

```
sensuctl handler update slack
```

Follow the prompts to add the `hourly` and `is_incident` filters to the Slackhandler.

Validating the filter

You can verify the proper behavior of this filter by using `sensu-backend` logs.The default location of these logs varies based on the platform used, but the troubleshooting guide provides this information.

Whenever an event is being handled, a log entry is added with the message `"handler":"slack","level":"debug","msg":"sending event to handler"`, followed bya second one with the message `"msg":"pipelined executed event pipeHandler","output":"","status":0`. However, if the event is being discarded byour filter, a log entry with the message `event filtered` will appear instead.

Next steps

You now know how to apply a filter to a handler and hopefully reduce alertfatigue. From this point, here are some recommended resources:

- Read the filters reference for in-depthdocumentation on filters.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Securing Sensu

As with any piece of software, it is critical to minimize any attack surface exposed by the software. Sensu is no different. The following component pieces need to be secured in order for Sensu to be considered production ready:

- etcd peer communication
- Backend API
- Dashboard
- Sensu agent to server communication

We'll cover securing each one of those pieces, starting with etcd peer communication.

Securing etcd peer communication

Let's start by covering how to secure etcd peer communication via the configuration at `/etc/sensu/backend.yml`. Let's look at the parameters you'll need to configure:

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

```
etcd-peer-client-cert-auth: "true"
etcd-peer-trusted-ca-file: "/path/to/your/peer/ca/file"
```

Securing the API and the dashboard

Let's go over how to secure the API and dashboard. Please note that by changing the parameters below, the server will now communicate over TLS and expect agents connecting to it to use the WebSocket secure protocol. In order for communication to continue, both this section and the following section must be completed.

Both the Sensu Go API and the dashboard use a common stanza in `/etc/sensu/backend.yml` to provide the certificate, key, and CA file needed to provide secure communication. Let's look at the attributes you'll need to configure:

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

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

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

In the example above, we provide the path to the cert, key and CA file. After restarting the `sensu-backend` service, the parameters are loaded and you are able to access the dashboard at https://localhost:3000. Configuring these attributes will also ensure that agents are able to communicate securely. Let's move on to securing agent to server communication.

Securing Sensu agent to server communication

We'll now discuss securing agent to server communication. Please note: by changing the agent configuration to communicate via WebSocket Secure protocol, the agent will no longer communicate over a plaintext connection. If the server is not secured as described in the section above, communication between the agent and server will not function.

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

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

In order to use WebSockets over SSL/TLS (wss), change the `backend-url` value to the `wss://` schema:

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

The agent will then connect Sensu servers over wss. Do note that by changing the configuration to wss, plaintext communication will not be possible.

It is also possible to provide a trusted CA as part of the agent configuration by passing `--trusted-ca-file` if starting the agent via `sensu-agent start`.

You may include it as part of the agent configuration in `/etc/sensu/agent.yml` as:

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

NOTE: If creating a Sensu cluster, every cluster member needs to be present in the configuration. See the Sensu Go clustering guide for more information on how to configure agents for a clustered configuration.

Hopefully you've found this useful! If you find any issues or have any questions, feel free to reach out in our Community Slack, or open an issue on Github.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

How to send alerts to Slack with handlers

What are Sensu handlers?

Sensu event handlers are actions executed by the Sensu server on events.

Why use a handler?

Handlers can be used for sending an email alert, creating or resolving an incident(in PagerDuty, for example), or storing metrics in a time-seriesdatabase (InfluxDB, for example).

Using a handler to send alerts to Slack

The purpose of this guide is to help you send alerts to Slack, on the channel `monitoring`, by configuring a handler named `slack` to a check named `check-cpu`. If you don't already have a check in place, this guide is agreat place to start.

Registering the asset

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins.In this guide, we'll use the Sensu Slack handler asset to power a `slack` handler.

You can use the following sensuctl example to register the Sensu Slack handler asset for Linux AMD64, or you can download the latest asset definition for your platform from Bonsai and register the asset using `sensuctl create --file filename.yml`.

```
sensuctl asset create sensu-slack-handler --url "https://github.com/sensu/sensu-sla
handler_1.0.3_linux_amd64.tar.gz" --sha512
"68720865127fbc7c2fe16ca4d7bbf2a187a2df703f4b4acae1c93e8a66556e9079e1270521999b5871
```

You should see a confirmation message from sensuctl.

```
Created
```

Getting a Slack webhook

If you're already an 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, start here to create a new workspace.)After saving, you'll see your webhook URL under Integration Settings.

Creating the handler

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

```
sensuctl handler create slack \
--type pipe \
--env-vars
"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T0000/B000/XXXXXXXX" \
--command "sensu-slack-handler --channel '#monitoring'" \
--runtime-assets sensu-slack-handler
```

You should see a confirmation message from sensuctl.

```
Created
```

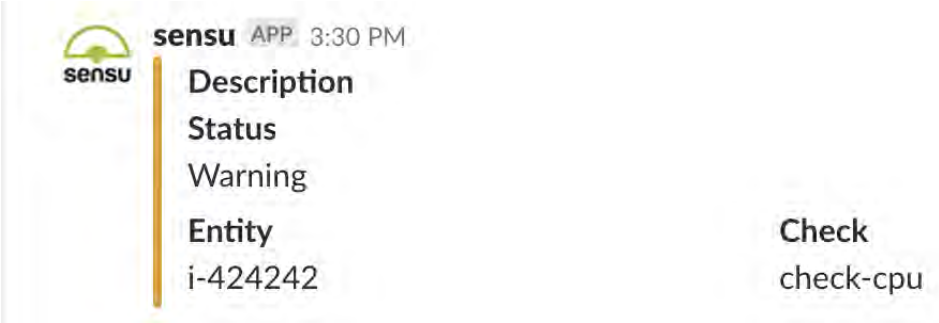
Assigning the handler to a check

With the `slack` handler now created, it can be assigned to a check. Here, sincewe want to receive Slack alerts whenever the CPU usage of our systems reach somespecific thresholds, we will apply our handler to the check `check-cpu`.

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

Validating the handler

It might take a few moments, once the handler is assigned to the check, for thecheck to be scheduled on the entities and the result sent back to Sensu backend,but once an event is handled, you should see the following message inSlack.



Otherwise, you can verify the proper behavior of this handler by using `sensu-backend` logs.See the troubleshooting guide 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"`, followedby a second one with the message `"msg":"pipelined executed event pipehandler","output":"","status":0`.

Next steps

You now know how to apply a handler to a check and take action on events. Fromthis point, here are some recommended resources:

- Read the handlers reference for in-depthdocumentation on handlers.
- Read our guide on reducing alert fatigue with filters.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Sensu service logging 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 walks you through how to add log forwarding from journald to syslog, have rsyslog write logging data to disk, and set up log rotation of the newly created log files.

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

```
ForwardToSyslog=yes
```

Next, set up rsyslog to write the logging data received from journald to `/var/log/sensu/servicename.log`. In this example, the `sensu-backend` and `sensu-agent` logging data is sent to individual files named after the service. The `sensu-backend` is not required if only setting up log forwarding for the `sensu-agent` service.

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

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

Restart rsyslog and journald to apply the new configuration:

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

Set up log rotation for newly created log files. This example rotates all log files in `/var/log/sensu/` weekly, unless the size of 100M is reached first. The last five rotated logs are kept and compressed, with the exception of the most recent one.

```
# Inside /etc/logrotate.d/sensu.conf
/var/log/sensu/* {
    weekly
    rotate 5
    size 100M
    compress
    delaycompress
}
```

You can use the following command to see what logrotate would do if it were executed now based on the above schedule and size threshold.

```
logrotate -d /etc/logrotate.d/sensu.conf
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Troubleshooting

Service logging

Logs produced by Sensu services – i.e. sensu-backend and sensu-agent – areoften the best place to start when troubleshooting a variety of issues.

Log levels

Each log message is associated with a log level, indicative of the relative severity of the event being logged:

Log level	Description
panic	Severe errors causing the service to shut down in an unexpected state
fatal	Fatal errors causing the service to shut down (status 0)
error	Non-fatal service error messages
warn	Warning messages indicating potential issues
info	Informational messages representing service actions
debug	Detailed service operation messages to help troubleshoot issues

These log levels can be configured by specifying the desired log level as thevalue of `log-level` in the service configuration file (e.g. `agent.yml` or `backend.yml` configuration files), or as an argument to the `--log-level` command line flag:

```
sensu-agent start --log-level debug
```

Changes to log level via configuration file or command line arguments requirerestarting the service. For guidance on restarting a service, pleaseconsult the Operating section of the agent or backend reference, respectively.

Log file locations

Sensu services print structured log messages to standard output.In order to capture these log messages to disk or another logging facility, Sensu servicesmake use of capabilities provided by the underlying operating system's servicemanagement. 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 sysvinit schemes. If you are running systemd as your service manager and would rather have logs written to `/var/log/sensu/`, see the guide to forwarding logs from journald to syslog.

In the table below, the common targets for logging and example commands forfollowing those logs are described. The name of the desired service, e.g. `backend` or `agent` may be substituted for `${service}` variable.

Platform	Version	Target	Command to follow log
RHEL/Centos	>= 7	journald	`journalctl --unit sensu-${service} --follow`
RHEL/Centos	<= 6	log file	`tail --follow /var/log/sensu/sensu-${service}`
Ubuntu	>= 15.04	journald	`journalctl --unit sensu-${service} --follow`

Ubuntu	<= 14.10	log file	`tail --follow /var/log/sensu/sensu-${service}`
Debian	>= 8	journald	`journalctl --unit sensu-${service} --follow`
Debian	<= 7	log file	`tail --follow /var/log/sensu/sensu-${service}`
Windows	Any	log file	`Get-Content - Path "C:\scripts\test.txt" -Wait`

NOTE: Platform versions described above are for reference only and do notsupercede the documented supported platforms.

Log messages

Permission issues

Files and folders within `/var/cache/sensu/` and `/var/lib/sensu/` need to be owned by the sensu user and group. You will see a logged error similar to the following 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"}
```

You can 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
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Dashboard

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Dashboard filtering

The Sensu dashboard supports filtering on the events, entities, checks, and silences pages.Dashboard filtering uses Sensu query expression syntax (for example: `entity.entity_class === "proxy"`) depending on the scope of the page.

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

Events page filtering

Filtering on the events page supports all entity and check attributes present in the event data, prefixed with `entity.` or `check.` respectively.

To show only events for the entity hostname `server1`:

```
entity.system.hostname === "server1"
```

To show only events with a warning or critical status produced by the check named `check_http`:

```
check.status > 0 && check.name === "check_http"
```

Entities page filtering

Filtering on the entities page assumes the entity scope and supports all entity attributes.

To show only entities of entity class `proxy`:

```
entity_class === "proxy"
```

To show only entities running on Linux or Windows:

```
system.os === "linux" || system.os === "windows"
```

Checks page filtering

Filtering on the check page assumes the check scope and supports all check attributes.

To show only the check named `check_cpu`:

```
name === "check_cpu"
```

To show only checks with the `publish` attribute set to `false`:

```
!publish
```

Silences page filtering

Filtering on the silences page assumes the silences scope and supports all silencing entry attributes.

To show only silences with the creator `admin`:

```
creator === "admin"
```

To show only silences applied to the check `check_cpu`:

```
check === "check_cpu"
```

Arrays

To filter based on an attribute that contains an array of elements, use the `.indexOf` method.

On the checks page, to show only checks with the handler `slack`:

```
handlers.indexOf("slack") >= 0
```

Regular expressions

The Sensu dashboard supports filtering with regular expressions using the `.match` syntax.

On the checks page, to show only checks with names prefixed with `metric-`:

```
!!name.match(/^metric-/)
```

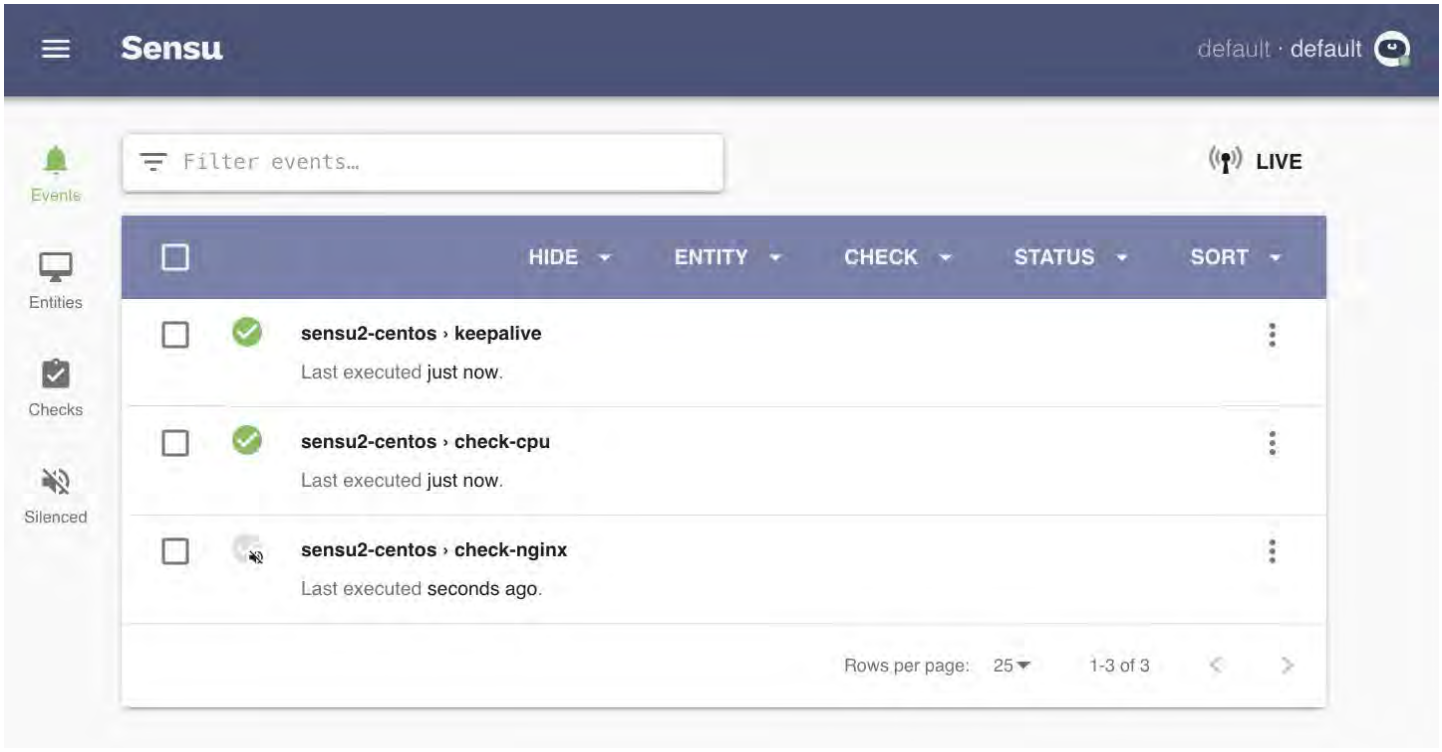
About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Dashboard overview

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



Accessing the dashboard

After starting the Sensu backend, you can access the dashboard in your browserby visiting http://localhost:3000. You may need to replace `localhost` with thehostname or IP address where

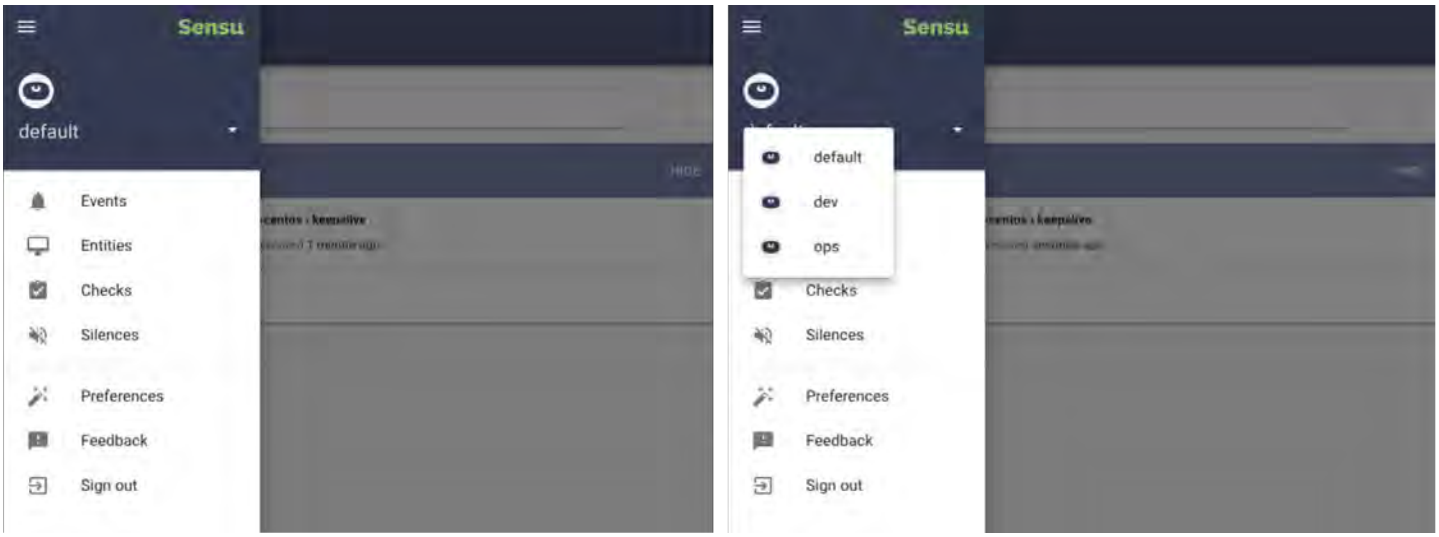
the Sensu backend is running.

Signing in

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

Namespaces

The dashboard displays events, entities, checks, and silences for a single namespace at a time.By default, the dashboard displays the `default` namespace.To switch namespaces, select the menu icon in the upper-left corner, and choose a namespace from the dropdown.



Sensu dashboard namespace switcher

Themes

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

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

API

- Users

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Assets API

The `/assets` API endpoint

`/assets` (GET)

The `/assets` API endpoint provides HTTP GET access to asset data.

EXAMPLE

The following example demonstrates a request to the `/assets` API, resulting ina JSON Array containing asset definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/assets -H "Authorization:
[
  {
    "url": "http://example.com/asset1.tar.gz",
    "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4
```

```
    "metadata": {
      "name": "check_script1",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  }
]
```

API Specification

/assets (GET)	
description	Returns the list of assets.
example url	http://hostname:8080/api/core/v2/namespaces/default/assets
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "url": "http://example.com/asset1.tar.gz",
    "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d1548
    "metadata": {
      "name": "check_script1",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  },
```

```
  {
    "url": "http://example.com/asset2.tar.gz",
    "sha512":
"37c9c85584c38ccf55a3ef3c2e8d154812246e5dda4a84f926bf4328fbad2b9cac873d1
    "metadata": {
      "name": "check_script2",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  }
]
```

`/assets` (POST)

/assets (POST)	
description	Create a Sensu asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets

payload

```
{
  "url": "http://example.com/asset1.tar.gz",
  "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d1548
  "metadata": {
    "name": "check_script1",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/assets/:asset` API endpoint

`/assets/:asset` (GET)

The `/assets/:asset` API endpoint provides HTTP GET access to asset data for specific `:asset` definitions, by asset `name`.

EXAMPLE

In the following example, querying the `/assets/:asset` API returns a JSON Mapcontaining the requested `:asset` definition (in this example: for the `:asset` named `check_script`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/assets/check_script -H "A
{
  "url": "http://example.com/asset.tar.gz",
  "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4
  "filters": [
    "system.os == 'linux'",
    "system.arch == 'amd64'"
  ],
  "metadata": {
    "name": "check_script",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```

API Specification

/assets/:asset (GET)	
description	Returns an asset.
example url	http://hostname:8080/api/core/v2/namespaces/default/assets/check_script
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

output	{
	"url": "http://example.com/asset.tar.gz",
	"sha512":
	"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e
	"filters": [
	"system.os == 'linux'",
	"system.arch == 'amd64'"
	],
	"metadata": {
	"name": "check_script",
	"namespace": "default",
	"labels": null,
	"annotations": null
	}
	}

`/assets/:asset` (PUT)

API Specification

/assets/:asset (PUT)

description	Create or update a Sensu asset.
example URL	http://hostname:8080/api/core/v2/namespaces/default/assets/check_script
payload	

```
{
  "url": "http://example.com/asset1.tar.gz",
  "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e
  "metadata": {
    "name": "check_script1",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Authorization API

The authorization API is available in Sensu Go version 5.1.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest API documentation.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Authentication providers API

The authentication providers API is available in Sensu Go version 5.2.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest API documentation.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Checks API

The `/checks` API endpoint

`/checks` (GET)

The `/checks` API endpoint provides HTTP GET access to check data.

EXAMPLE

The following example demonstrates a request to the `/checks` API, resulting ina JSON Array containing check definitions.

```
curl -H "Authorization: Bearer $SENSU_TOKEN"
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"
    }
  }
]
```

API Specification

/checks (GET)	
description	Returns the list of checks.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)

output


```
[
  {
    "command": "check-cpu.sh -w 75 -c 90",
    "handlers": [
      "slack"
    ],
    "interval": 60,
    "publish": true,
    "subscriptions": [
      "linux"
    ],
    "metadata": {
      "name": "check-cpu",
      "namespace": "default"
    }
  },
  {
    "command": "http_check.sh https://sensu.io",
    "handlers": [
      "slack"
    ],
    "interval": 15,
    "proxy_entity_name": "sensu.io",
    "publish": true,
    "subscriptions": [
      "site"
    ],
    "metadata": {
      "name": "check-sensu-site",
      "namespace": "default"
    }
  }
]
```


`/checks` (POST)

EXAMPLE

In the following example, an HTTP POST request is submitted to the `/checks` API to create a `check-cpu` check.The request includes the check definition in the request body and returns a successful HTTP 200 OK response and the created check definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "command": "check-cpu.sh -w 75 -c 90",
  "subscriptions": [
    "linux"
  ],
  "interval": 60,
  "publish": true,
  "handlers": [
    "slack"
  ],
  "metadata": {
    "name": "check-cpu",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks

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

```
    "name": "check-cpu",
    "namespace": "default"
  }
}
```

API Specification

/checks (POST)

description	Create a Sensu check.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks

example payload	<pre>{
	  "command": "check-cpu.sh -w 75 -c 90",
	  "subscriptions": [
	  "linux"
	  ],
	  "interval": 60,
	  "publish": true,
	  "handlers": [
	  "slack"
	  ],
	  "metadata": {
	  "name": "check-cpu",
	  "namespace": "default"
	  }
	}</pre>

payload parameters	Required check attributes: `interval` (integer) or `cron` (string), and a `metadata` scope containing `name` (string) and `namespace` (string). For more information about creating checks, see the check reference.
response codes	**Success**: 200 (OK) **Malformed**: 400 (Bad Request) **Error**: 500 (Internal Server Error)

The `/checks/:check` API endpoint

`/checks/:check` (GET)

The `/checks/:check` API endpoint provides HTTP GET access to check data for specific `:check` definitions, by check `name`.

EXAMPLE

In the following example, querying the `/checks/:check` API returns a JSON Mapcontaining the requested `:check` definition (in this example: for the `:check` named `check-cpu`).

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu

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"
  }
}
```

API Specification

/checks/:check (GET)	
description	Returns a 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"
  ],
  "interval": 60,
  "publish": true,
  "subscriptions": [
    "linux"
  ],
  "metadata": {
    "name": "check-cpu",
    "namespace": "default"
  }
}
```

`/checks/:check` (PUT)

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/checks/:check` API to update the `check-cpu` check, resulting in a 200 (OK) HTTP response code and the updated check definition.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "command": "check-cpu.sh -w 75 -c 90",
  "handlers": [
    "slack"
  ],
  "interval": 60,
  "publish": true,
  "subscriptions": [
    "linux"
  ],
  "metadata": {
    "name": "check-cpu",
    "namespace": "default"
  }
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu

HTTP/1.1 200 OK
```

API Specification

/checks/:check (PUT)	
description	Create or update a Sensu check given the name of the check as a URL parameter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/check-cpu
payload	

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

payload parameters	Required check attributes: `interval` (integer) or `cron` (string), and a `metadata` scope containing `name` (string) and `namespace` (string). For more information about creating checks, see the check reference.
response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/checks/:check` (DELETE)

The `/checks/:check` API endpoint provides HTTP DELETE access to delete a check from Sensu given the check name.

EXAMPLE

The following example shows a request to delete the check named `check-cpu`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
```

```
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu

HTTP/1.1 204 No Content
```

API Specification

/checks/:check (DELETE)	
description	Removes a check from Sensu given the check name.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check-cpu
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

The `/checks/:check/execute` API endpoint

`/checks/:check/execute` (POST)

The `/checks/:check/execute` API endpoint provides HTTP POST access to create an ad-hoc check execution request, allowing you to execute a check on demand.

EXAMPLE

In the following example, an HTTP POST request is submitted to the `/checks/:check/execute` API to execute the `check-sensu-site` check.The request includes the check name in the request body and returns a successful HTTP 202 Accepted response and an `issued` timestamp.

```
curl -X POST \
```

```
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{"check": "check-sensu-site"}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-sensu-
site/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 adhoc request to execute a check given the check name.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/check-sensu-site/execute

payload	<pre>{
	  "check": "check-sensu-site",
	  "subscriptions": [
	  "entity:i-424242"
	  ]
	}</pre>

payload parameters	`check` (required): the name of the check to execute, and `subscriptions` (optional): 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**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/checks/:check/hooks/:type` API endpoint

`/checks/:check/hooks/:type` (PUT)

The `/checks/:check/hooks/:type` API endpoint provides HTTP PUT access to assign a hook to a check.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/checks/:check/hooks/:type` API,assigning the `process_tree` hook to the `check-cpu` check in the event of a `critical` type check result, resulting in a successful 204 (No Content) HTTP response code.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "critical": [
    "process_tree"
  ]
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-cpu/hooks/critical

HTTP/1.1 204 No Content
```

API Specification

checks/:check/hooks/:type

(PUT)	
description	Assigns a hook to a check given the check name and check resp type.
example URL	http://hostname:8080/api/core/v2/namespaces/default/checks/ch cpu/hooks/critical
example payload	{ "critical": ["example-hook1", "example-hook2"] }
payload parameters	This endpoint requires a JSON map of check response types (for example: `critical`, `warning`), each containing an array of hoo names.
response codes	**Success**: 204 (No Content) **Malformed**: 400 (Bad Request) **Error**: 500 (Internal Server Error)

The `/checks/:check/hooks/:type/hook/:hook` API endpoint

`/checks/:check/hooks/:type/hook/:hook` (DELETE)

This endpoint provides HTTP DELETE access to a remove a hook from a check.

EXAMPLE

The following example shows a request to remove the `process_tree` hook from the `check-cpu` check, resulting in a successful 204 (No Content) HTTP response code.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/checks/check-
cpu/hooks/critical/hook/process_tree

HTTP/1.1 204 No Content
```

API Specification

/checks/:check/hooks/:type/hook/:hook (DELETE)	
description	Removes a single hook from a check given the check name, check response type, and hook name. See the checks reference for available types.
example url	http://hostname:8080/api/core/v2/namespaces/default/checks/check-cpu/hooks/critical/hook/process_tree
response codes	**Success**: 204 (No Content) **Missing**: 404 (Not Found) **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Cluster role bindings API

The `/clusterrolebindings` API endpoint

`/clusterrolebindings` (GET)

The `/clusterrolebindings` API endpoint provides HTTP GET access to cluster role binding data.

EXAMPLE

The following example demonstrates a request to the `/clusterrolebindings` API, resulting ina JSON Array containing cluster role binding definitions.

```
curl http://127.0.0.1:8080/api/core/v2/clusterrolebindings -H "Authorization:
Bearer $SENSU_TOKEN"
[
  {
    "subjects": [
```

```
        {
          "type": "Group",
          "name": "cluster-admins"
        }
      ],
      "role_ref": {
        "type": "ClusterRole",
        "name": "cluster-admin"
      },
      "metadata": {
        "name": "cluster-admin"
      }
    },
    {
      "subjects": [
        {
          "type": "Group",
          "name": "system:agents"
        }
      ],
      "role_ref": {
        "type": "ClusterRole",
        "name": "system:agent"
      },
      "metadata": {
        "name": "system:agent"
      }
    }
]
```


API Specification

/clusterrolebindings (GET)	
description	Returns the list of cluster role bindings.
example url	http://hostname:8080/api/core/v2/clusterrolebindings

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"
    }
  }
]
```


`/clusterrolebindings` (POST)

The `/clusterrolebindings` API endpoint provides HTTP POST access to create a cluster role binding.

EXAMPLE

In the following example, an HTTP POST request is submitted to the `/clusterrolebindings` API to create a cluster role binding that assigns the `cluster-admin` cluster role to the user `bob`. The request includes the cluster role binding definition in the request body and returns a successful HTTP 200 OK response and the created cluster role binding definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}
```

API Specification

/clusterrolebindings (POST)	
description	Create a Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebindings
payload	`{ "subjects": [ { "type": "User", "name": "bob" } ], "role_ref": { "type": "ClusterRole", "name": "cluster-admin" }, "metadata": { "name": "bob-binder" } }`
response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

The `/clusterrolebindings/:clusterrolebinding` API endpoint

`/clusterrolebindings/:clusterrolebinding` (GET)

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP GET access to

cluster role binding data for specific `:clusterrolebinding` definitions, by cluster role binding `name`.

EXAMPLE

In the following example, querying the `/clusterrolebindings/:clusterrolebinding` API returns a JSON Mapcontaining the requested `:clusterrolebinding` definition (in this example: for the `:clusterrolebinding` named `bob-binder`).

```
curl http://127.0.0.1:8080/api/core/v2/clusterrolebindings/bob-binder -H
"Authorization: Bearer $SENSU_TOKEN"

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "User",
      "name": "bob"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "bob-binder"
  }
}
```

API Specification

/clusterrolebindings/:clusterrolebinding (GET)	
description	Returns a cluster role binding.
example url	http://hostname:8080/api/core/v2/clusterrolebind

	binder
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

output	{
	"subjects": [
	{
	"type": "User",
	"name": "bob"
	}
	],
	"role_ref": {
	"type": "ClusterRole",
	"name": "cluster-admin"
	},
	"metadata": {
	"name": "bob-binder"
	}
	}

`/clusterrolebindings/:clusterrolebinding` (PUT)

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP PUT access to create or update a cluster role binding, by cluster role binding `name`.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/clusterrolebindings/:clusterrolebinding` API to create a cluster role binding that assigns the `cluster-admin` cluster role to users in the group `ops`.The request includes the cluster role binding definition in the request body and returns a successful HTTP 200 OK response and the created

cluster role binding definition.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "subjects": [
    {
      "type": "Group",
      "name": "ops"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "ops-group-binder"
  }
}' \
http://127.0.0.1:8080/api/core/v2/clusterrolebindings/ops-group-binder

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "Group",
      "name": "ops"
    }
  ],
  "role_ref": {
    "type": "ClusterRole",
    "name": "cluster-admin"
  },
  "metadata": {
    "name": "ops-group-binder"
  }
}
```

API Specification

/clusterrolebindings/:clusterrolebinding (PUT)	
description	Create or update a Sensu cluster role binding.
example URL	http://hostname:8080/api/core/v2/clusterrolebind group-binder

payload	{
	"subjects": [
	{
	"type": "Group",
	"name": "ops"
	}
	],
	"role_ref": {
	"type": "ClusterRole",
	"name": "cluster-admin"
	},
	"metadata": {
	"name": "ops-group-binder"
	}
	}

response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/clusterrolebindings/:clusterrolebinding` (DELETE)

The `/clusterrolebindings/:clusterrolebinding` API endpoint provides HTTP DELETE access to delete a cluster role binding from Sensu given the cluster role binding name.

EXAMPLE

The following example shows a request to delete the cluster role binding `ops-binding`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
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 given cluster role binding name.
example url	http://hostname:8080/api/core/v2/clusterrolebinding
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Cluster roles API

The `/clusterroles` API endpoint

`/clusterroles` (GET)

The `/clusterroles` API endpoint provides HTTP GET access to cluster role data.

EXAMPLE

The following example demonstrates a request to the `/clusterroles` API, resulting ina JSON Array containing cluster role definitions.

```
curl http://127.0.0.1:8080/api/core/v2/clusterroles -H "Authorization: Bearer
$SENSU_TOKEN"

HTTP/1.1 200 OK
[
```

```
{
  "rules": [
    {
      "verbs": [
        "*"
      ],
      "resources": [
        "assets",
        "checks",
        "entities",
        "extensions",
        "events",
        "filters",
        "handlers",
        "hooks",
        "mutators",
        "silenced",
        "roles",
        "rolebindings"
      ],
      "resource_names": null
    },
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "namespaces"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "admin"
  }
},
{
  "rules": [
    {
```

```
        "verbs": [
          "*"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "cluster-admin"
    }
  }
]
```


API Specification

/clusterroles (GET)	
description	Returns the list of cluster roles.
example url	http://hostname:8080/api/core/v2/clusterroles
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "rules": [
      {
        "verbs": [
          "*"
        ],
        "resources": [
```

```
            "*"
          ],
          "resource_names": null
        }
      ],
      "metadata": {
        "name": "cluster-admin"
      }
    }
]
```

`/clusterroles` (POST)

/clusterroles (POST)	
description	Create a Sensu cluster role.
example URL	http://hostname:8080/api/core/v2/clusterroles
payload	

```
{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": null
    }
  ]
```

```
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/clusterroles/:clusterrole` API endpoint

`/clusterroles/:clusterrole` (GET)

The `/clusterroles/:clusterrole` API endpoint provides HTTP GET access to cluster role data for specific `:clusterrole` definitions, by cluster role `name`.

EXAMPLE

In the following example, querying the `/clusterroles/:clusterrole` API returns a JSON Map containing the requested `:clusterrole` definition (in this example: for the `:clusterrole` named `global-event-reader`).

```
curl http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-reader -H
"Authorization: Bearer $SENSU_TOKEN"

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "global-event-reader"
  },
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
```

```
        "events"
      ],
      "resource_names": null
    }
  ]
}
```

API Specification

/clusterroles/:clusterrole (GET)	
description	Returns a cluster role.
example url	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

output	{
	"metadata": {
	"name": "global-event-reader"
	},
	"rules": [
	{
	"verbs": [
	"get",
	"list"
	],
	"resources": [
	"events"
	],
	"resource_names": **null**

```
    }
  ]
}
```

`/clusterroles/:clusterrole` (PUT)

API Specification

/clusterroles/:clusterrole (PUT)	
description	Create or update a 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)

`/clusterroles/:clusterrole` (DELETE)

The `/clusterroles/:clusterrole` API endpoint provides HTTP DELETE access to delete a cluster role from Sensu given the cluster role name.

EXAMPLE

The following example shows a request to delete the cluster role `global-event-reader`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/clusterroles/global-event-reader

HTTP/1.1 204 No Content
```

API Specification

/clusterroles/:clusterrole (DELETE)	
description	Removes a cluster role from Sensu given the cluster role name.
example url	http://hostname:8080/api/core/v2/clusterroles/global-event-reader
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Cluster API

The `/cluster/members` API endpoint

`/cluster/members` (GET)

The `/cluster/members` API endpoint provides HTTP GET access to Sensu cluster data.

EXAMPLE

The following example demonstrates a request to the `/cluster/members` API, resulting ina JSON Map containing a Sensu cluster definition.

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/cluster/members

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 cluster definition.
example url	http://hostname:8080/api/core/v2/cluster/members
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
example output	


```
{
  "header": {
    "cluster_id": 4255616304056076734,
    "member_id": 9882886658148554927,
    "raft_term": 2
```

```
  },
  "members": [
    {
      "ID": 9882886658148554927,
      "name": "default",
      "peerURLs": [
        "http://127.0.0.1:2380"
      ],
      "clientURLs": [
        "http://127.0.0.1:2379"
      ]
    }
  ]
}
```

`/cluster/members` (POST)

The `/cluster/members` API endpoint provides HTTP POST access to create a Sensu cluster member.

EXAMPLE

```
curl -X POST \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/cluster/members?peer-
addrs=http://127.0.0.1:2380

HTTP/1.1 200 OK
{
  "header": {
    "cluster_id": 4255616304056077000,
    "member_id": 9882886658148555000,
    "raft_term": 2
  },
  "members": [
    {
```

```
      "ID": 9882886658148555000,
      "name": "default",
      "peerURLs": [
        "http://127.0.0.1:2380"
      ],
      "clientURLs": [
        "http://localhost:2379"
      ]
    }
  ]
}
```


API Specification

/cluster/members/:member (POST)	
description	Creates a cluster member.
example url	http://hostname:8080/api/core/v2/cluster/members?peer-addrs=http://127.0.0.1:2380
query parameters	☐ `peer-addrs` (required): A comma-delimited list of peer addresses
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

The `/cluster/members/:member` API endpoint

`/cluster/members/:member` (PUT)

EXAMPLE

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/cluster/members/8927110dc66458af?peer-
addrs=http://127.0.0.1:2380

HTTP/1.1 200 OK
{
  "header": {
    "cluster_id": 4255616304056077000,
    "member_id": 9882886658148555000,
    "raft_term": 2
  },
  "members": [
    {
      "ID": 9882886658148555000,
      "name": "default",
      "peerURLs": [
        "http://127.0.0.1:2380"
      ],
      "clientURLs": [
        "http://localhost:2379"
      ]
    }
  ]
}
```

API Specification

/cluster/members/:member (PUT)	
description	Creates a cluster member.
example url	http://hostname:8080/api/core/v2/cluster/members/8927110dc6 peer-addrs=http://127.0.0.1:2380

url parameters	☐ `8927110dc66458af` (required): Required hex-encoded ui cluster member ID generated using `sensuctl cluster member-list`
query parameters	☐ `peer-addrs` (required): A comma-delimited list of peer addresses
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

`/cluster/members/:member` (DELETE)

The `/cluster/members/:member` API endpoint provides HTTP DELETE access to remove a Sensu cluster member.

EXAMPLE

The following example shows a request to remove the Sensu cluster member with the ID `8927110dc66458af`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/cluster/members/8927110dc66458

HTTP/1.1 204 No Content
```

API Specification

/cluster/members/:member (DELETE)

description	Removes a member from a Sensu cluster given 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**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Entities API

The `/entities` API endpoint

`/entities` (GET)

The `/entities` API endpoint provides HTTP GET access to entity data.

EXAMPLE

The following example demonstrates a request to the `/entities` API, resulting ina JSON Array containing entity definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/entities -H
"Authorization: Bearer $SENSU_TOKEN"
[
  {
    "entity_class": "agent",
```

```
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
              "10.0.2.15/24",
              "fe80::f50c:b029:30a5:3e26/64"
            ]
          },
          {
            "name": "enp0s8",
            "mac": "08:00:27:9f:5d:f3",
            "addresses": [
              "172.28.128.3/24",
              "fe80::a00:27ff:fe9f:5df3/64"
            ]
          }
        ]
      },
      "arch": "amd64"
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
    "deregister": false,
```

```
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  }
]
```

API Specification

/entities (GET)	
description	Returns the list of entities.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)
output	

```
[
  {
```

```
    "entity_class": "agent",
    "system": {
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
              "10.0.2.15/24",
              "fe80::f50c:b029:30a5:3e26/64"
            ]
          },
          {
            "name": "enp0s8",
            "mac": "08:00:27:9f:5d:f3",
            "addresses": [
              "172.28.128.3/24",
              "fe80::a00:27ff:fe9f:5df3/64"
            ]
          }
        ]
      },
      "arch": "amd64"
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
```

```
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
  }
]
```

`/entities` (POST)

/entities (POST)	
description	Create a Sensu entity.
example URL	http://hostname:8080/api/core/v2/namespaces/default/entities
payload	

```
{
  "entity_class": "proxy",
  "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)

The `/entities/:entity` API endpoint

`/entities/:entity` (GET)

The `/entities/:entity` API endpoint provides HTTP GET access to entity data for specific `:entity` definitions, by entity `name`.

EXAMPLE

In the following example, querying the `/entities/:entity` API returns a JSON Mapcontaining the requested `:entity` definition (in this example: for the `:entity` named `sensu-centos`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/sensu-centos
-H "Authorization: Bearer $SENSU_TOKEN"
{
  "entity_class": "agent",
  "system": {
    "hostname": "sensu-centos",
    "os": "linux",
```

```
    "platform": "centos",
    "platform_family": "rhel",
    "platform_version": "7.4.1708",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "enp0s3",
          "mac": "08:00:27:11:ad:d2",
          "addresses": [
            "10.0.2.15/24",
            "fe80::f50c:b029:30a5:3e26/64"
          ]
        },
        {
          "name": "enp0s8",
          "mac": "08:00:27:9f:5d:f3",
          "addresses": [
            "172.28.128.3/24",
            "fe80::a00:27ff:fe9f:5df3/64"
          ]
        }
      ]
    },
    "arch": "amd64"
  },
  "subscriptions": [
    "entity:sensu-centos"
  ],
  "last_seen": 1543349936,
  "deregister": false,
  "deregistration": {},
  "user": "agent",
  "redact": [
```

```
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default",
      "labels": null,
      "annotations": null
    }
}
```

API Specification

/entities/:entity (GET)	
description	Returns a entity.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
response type	Map
response codes	**Success**: 200 (OK) **Missing**: 404 (Not Found) **Error**: 500 (Internal Server Error)
output	

```
{
  "entity_class": "agent",
  "system": {
```

```
      "hostname": "sensu-centos",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
              "10.0.2.15/24",
              "fe80::f50c:b029:30a5:3e26/64"
            ]
          },
          {
            "name": "enp0s8",
            "mac": "08:00:27:9f:5d:f3",
            "addresses": [
              "172.28.128.3/24",
              "fe80::a00:27ff:fe9f:5df3/64"
            ]
          }
        ]
      },
      "arch": "amd64"
    },
    "subscriptions": [
      "entity:sensu-centos"
    ],
    "last_seen": 1543349936,
    "deregister": false,
    "deregistration": {},
```

```
  "user": "agent",
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
    "secret"
  ],
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```

`/entities/:entity` (PUT)

API Specification

/entities/:entity (PUT)	
description	Create or update a Sensu entity.
example URL	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
payload	

```
{
  "entity_class": "proxy",
  "subscriptions": [
```

```
    "web"
  ],
  "deregister": false,
  "deregistration": {},
  "metadata": {
    "name": "sensu-centos",
    "namespace": "default",
    "labels": null,
    "annotations": null
  }
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

`/entities/:entity` (DELETE)

The `/entities/:entity` API endpoint provides HTTP DELETE access to delete an entity from Sensu given the entity name.

EXAMPLE

The following example shows a request to delete the entity `server1`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/entities/server1

HTTP/1.1 204 No Content
```

API Specification

/entities/:entity (DELETE)	
description	Removes a entity from Sensu given the entity name.
example url	http://hostname:8080/api/core/v2/namespaces/default/entities/sensu-centos
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

For Nagios

Pricing

Events API

The `/events` API endpoint

`/events` (GET)

The `/events` API endpoint provides HTTP GET access to event data.

EXAMPLE

The following example demonstrates a request to the `/events` API, resulting ina JSON Array containing event definitions.

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events
```

```
HTTP/1.1 200 OK
[
  {
    "timestamp": 1542667666,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "testing",
        "entity:webserver01"
      ],
      "metadata": {
        "name": "check-nginx",
        "namespace": "default",
        "labels": null,
        "annotations": null
      }
    },
    "check": {
      "check_hooks": null,
      "duration": 2.033888684,
      "command": "http_check.sh http://localhost:80",
      "handlers": [
        "slack"
      ],
      "high_flap_threshold": 0,
      "interval": 20,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [],
      "subscriptions": [
        "testing"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
```

```
        "stdin": false,
        "ttl": 0,
        "timeout": 0,
        "duration": 0.010849143,
        "output": "",
        "state": "failing",
        "status": 1,
        "total_state_change": 0,
        "last_ok": 0,
        "occurrences": 1,
        "occurrences_watermark": 1,
        "output_metric_format": "",
        "output_metric_handlers": [],
        "env_vars": null,
        "metadata": {
          "name": "check-nginx",
          "namespace": "default",
          "labels": null,
          "annotations": null
        }
      }
    }
]
```


API Specification

/events (GET)	
description	Returns the list of events.
example url	http://hostname:8080/api/core/v2/namespaces/default/events
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "timestamp": 1542667666,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "testing",
        "entity:webserver01"
      ],
      "metadata": {
        "name": "check-nginx",
        "namespace": "default",
        "labels": null,
        "annotations": null
      }
    },
    "check": {
      "check_hooks": null,
      "duration": 2.033888684,
      "command": "http_check.sh http://localhost:80",
      "handlers": [
        "slack"
      ],
      "high_flap_threshold": 0,
      "interval": 20,
      "low_flap_threshold": 0,
      "publish": true,
      "runtime_assets": [],
      "subscriptions": [
        "testing"
      ],
      "proxy_entity_name": "",
      "check_hooks": null,
      "stdin": false,
      "ttl": 0,
```

```
        "timeout": 0,
        "duration": 0.010849143,
        "output": "",
        "state": "failing",
        "status": 1,
        "total_state_change": 0,
        "last_ok": 0,
        "occurrences": 1,
        "occurrences_watermark": 1,
        "output_metric_format": "",
        "output_metric_handlers": [],
        "env_vars": null,
        "metadata": {
          "name": "check-nginx",
          "namespace": "default",
          "labels": null,
          "annotations": null
        }
      }
    }
]
```


`/events` (POST)

The `/events` API endpoint provides HTTP POST access to create an event and send it to the Sensu pipeline.

EXAMPLE

In the following example, an HTTP POST request is submitted to the `/events` API to create an event. The request includes information about the check and entity represented by the event and returns a successful HTTP 200 OK response and the event definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_TOKEN" \
```

```
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "state": "failing",
    "status": 2,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "timestamp": 1552582569
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events

HTTP/1.1 200 OK
{"timestamp":1552582569,"entity":{"entity_class":"proxy","system":{"network":{"inte
error","state":"failing","status":2,"total_state_change":0,"last_ok":0,"occurrences
```

API Specification

/events (POST)

description	Create a Sensu event for a new entity and check combination. To create an event for an existing entity and check combination or to update an existing event, use the `/events/:entity/:check` PUT endpoint.
example URL	http://hostname:8080/api/core/v2/namespaces/default/events
payload	

```
{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "state": "failing",
    "status": 2,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "timestamp": 1552582569
}
```

payload parameters	See the payload parameters section for the `/events/:entity/:check` PUT endpoint.
response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Conflict**: 409 (Event already exists for the entity and check) ☐ **Error**: 500 (Internal Server Error)

The `/events/:entity` API endpoint

`/events/:entity` (GET)

The `/events/:entity` API endpoint provides HTTP GET access to event data specific to an `:entity`, by entity `name`.

EXAMPLE

In the following example, querying the `/events/:entity` API returns a list of Sensu events for the `sensu-go-sandbox` entity and a successful HTTP 200 OK response.

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox

HTTP/1.1 200 OK
[
  {
    "timestamp": 1543871497,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "linux",
        "entity:sensu-go-sandbox"
      ],
      "last_seen": 1543858763,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      }
    },
    "check": {
      "command": "check-cpu.sh -w 75 -c 90",
      "duration": 1.054253257,
      "executed": 1543871496,
      "history": [
        {
          "status": 0,
          "executed": 1543870296
        }
```

```
      ],
      "issued": 1543871496,
      "output": "CPU OK - Usage:.50\n",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1543871497,
      "occurrences": 1,
      "metadata": {
        "name": "check-cpu",
        "namespace": "default"
      }
    },
    "metadata": {
      "namespace": "default"
    }
  },
  {
    "timestamp": 1543871524,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "linux",
        "entity:sensu-go-sandbox"
      ],
      "last_seen": 1543871523,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      }
    },
    "check": {
      "handlers": [
        "keepalive"
      ],
```

```
      "executed": 1543871524,
      "history": [
        {
          "status": 0,
          "executed": 1543871124
        }
      ],
      "issued": 1543871524,
      "output": "",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1543871524,
      "occurrences": 1,
      "metadata": {
        "name": "keepalive",
        "namespace": "default"
      }
    },
    "metadata": {}
  }
]
```


API Specification

/events/:entity (GET)	
description	Returns a list of events for the specified entity.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox
response type	Array
response codes	**Success**: 200 (OK) **Missing**: 404 (Not Found) **Error**: 500 (Internal Server Error)

output


```
[
  {
    "timestamp": 1543871524,
    "entity": {
      "entity_class": "agent",
      "system": {
        "hostname": "webserver01",
        "...": "...",
        "arch": "amd64"
      },
      "subscriptions": [
        "linux",
        "entity:sensu-go-sandbox"
      ],
      "last_seen": 1543871523,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      }
    },
    "check": {
      "handlers": [
        "keepalive"
      ],
      "executed": 1543871524,
      "history": [
        {
          "status": 0,
          "executed": 1543871124
        }
      ],
      "issued": 1543871524,
      "output": "",
      "state": "passing",
      "status": 0,
      "total_state_change": 0,
      "last_ok": 1543871524,
      "occurrences": 1,
      "metadata": {
```

```
        "name": "keepalive",
        "namespace": "default"
      }
    },
    "metadata": {}
  }
]
```


The `/events/:entity/:check` API endpoint

`/events/:entity/:check` (GET)

API Specification

/events/:entity/:check (GET)	
description	Returns an event for a given entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox/check-cpu
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)
output	`{ "timestamp": 1543871524, "entity": { "entity_class": "agent", "system": { "hostname": "webserver01",`

```
      "...": "...",
      "arch": "amd64"
    },
    "subscriptions": [
      "linux",
      "entity:sensu-go-sandbox"
    ],
    "last_seen": 1543871523,
    "metadata": {
      "name": "sensu-go-sandbox",
      "namespace": "default"
    }
  },
  "check": {
    "handlers": [
      "keepalive"
    ],
    "executed": 1543871524,
    "history": [
      {
        "status": 0,
        "executed": 1543871124
      }
    ],
    "issued": 1543871524,
    "output": "",
    "state": "passing",
    "status": 0,
    "total_state_change": 0,
    "last_ok": 1543871524,
    "occurrences": 1,
    "metadata": {
      "name": "keepalive",
      "namespace": "default"
    }
  },
  "metadata": {}
}
```

`/events/:entity/:check` (PUT)

The `/events/:entity/:check` API endpoint provides HTTP PUT access to create or update an event and send it to the Sensu pipeline.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/events/:entity/:check` API to create an event for the `server1` entity and the `server-health` check and process it using the `slack` event handler.The event includes a status code of `1`, indicating a warning, and an output message of "Server error".

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "timestamp": 1552582569
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-
health
```

The request returns a 200 (OK) HTTP response code and the resulting event definition.

```
HTTP/1.1 200 OK
{"timestamp":1552582569,"entity":{"entity_class":"proxy","system":{"network":{"inte
error","status":1,"total_state_change":0,"last_ok":0,"occurrences":0,"occurrences_w
```

You can use sensuctl or the Sensu dashboard to see the event.

```
sensuctl event list
```

You should see the event with the status and output specified in the request.

```
Entity    Check           Output          Status  Silenced       Timestamp
─────────────────────────────────────────────────────────────────────────────

server1   server-health   Server error       1      false      2019-03-14 16:56:09 +0000 UTC
```

API Specification

/events/:entity/:check (PUT)	
description	Creates an event for a given entity and check.
example url	http://hostname:8080/api/core/v2/namespaces/default/events/server1/s 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"
    }
  },
  "timestamp": 1552582569
}
```

payload parameters	See the payload parameters section below.
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

Payload parameters

The `/events/:entity/:check` PUT endpoint requires a request payload containing an `entity` scope and a `check` scope.The `entity` scope contains information about the component of your infrastructure represented by the event.At a minimum, Sensu requires the `entity` scope to contain the `entity_class` (`agent` or `proxy`) and the entity `name` and `namespace` within a `metadata` scope.For more information about entity attributes, see the entity specification.

The `check` scope contains information about the event status and how the event was created.At a minimum, Sensu requires the `check` scope to contain a `name` within a `metadata` scope and either an `interval` or `cron` attribute.For more information about check attributes, see the check specification.

Example request with minimum required event attributes

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "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 shown above let you create an event using the `/events/:entity/:check` PUT endpoint, however the request can include any attributes defined in the event specification. To create useful, actionable events, we recommend adding check attributes such as the event `status` (`0` for OK, `1` for warning, `2` for critical), an `output` message, and one or more event `handlers`. For more information about these attributes and their available values, see the event specification.

While a `timestamp` is not required to create an event, Sensu assigns a timestamp of `0` (January 1, 1970) to events without a specified timestamp, so we recommend adding a Unix timestamp when creating events.

Example request with minimum recommended event attributes

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
```

```
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
  },
  "check": {
    "output": "Server error",
    "status": 1,
    "handlers": ["slack"],
    "interval": 60,
    "metadata": {
      "name": "server-health"
    }
  },
  "timestamp": 1552582569
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-
health
```

Creating metric events

In addition to the `entity` and `check` scopes, Sensu events can include a `metrics` scope containing metrics in Sensu metric format.See the events reference and for more information about Sensu metric format.

Example request including metrics

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "entity": {
    "entity_class": "proxy",
    "metadata": {
      "name": "server1",
      "namespace": "default"
    }
```

```
  },
  "check": {
    "status": 0,
    "output_metric_handlers": ["influxdb"],
    "interval": 60,
    "metadata": {
      "name": "server-metrics"
    }
  },
  "metrics": {
    "handlers": [
      "influxdb"
    ],
    "points": [
      {
        "name": "server1.server-metrics.time_total",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "server1.server-metrics.time_namelookup",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.004
      }
    ]
  },
  "timestamp": 1552582569
}' \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/server1/server-
metrics
```

`/events/:entity/:check` (DELETE)

EXAMPLE

The following example shows a request to delete the event produced by the `sensu-go-sandbox`

entity and `check-cpu` check, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox/check-cpu

HTTP/1.1 204 No Content
```

API Specification

/events/:entity/:check (DELETE)	
description	Deletes the event created by the specified entity using the specified check
example url	http://hostname:8080/api/core/v2/namespaces/default/events/sensu-go-sandbox/check-cpu
response codes	☐ **Success**: 204 (No Content) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Filters API

The `/filters` API endpoint

`/filters` (GET)

The `/filters` API endpoint provides HTTP GET access to filter data.

EXAMPLE

The following example demonstrates a request to the `/filters` API, resulting ina JSON Array containing filter definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/filters -H
"Authorization: Bearer $SENSU_TOKEN"
[
  {
    "metadata": {
```

```
      "name": "state_change_only",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ],
    "runtime_assets": []
  }
]
```


API Specification

/filters (GET)	
description	Returns the list of filters.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "metadata": {
      "name": "state_change_only",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
```

```
    ],
    "runtime_assets": []
  },
  {
    "metadata": {
      "name": "development_filter",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "action": "deny",
    "expressions": [
      "event.entity.metadata.namespace == 'production'"
    ],
    "runtime_assets": []
  }
]
```

`/filters` (POST)

/filters (POST)	
description	Create a Sensu 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 == 'production'"
```

```
  ],
  "runtime_assets": []
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/filters/:filter` API endpoint

`/filters/:filter` (GET)

The `/filters/:filter` API endpoint provides HTTP GET access to filter data for specific `:filter` definitions, by filter `name`.

EXAMPLE

In the following example, querying the `/filters/:filter` API returns a JSON Mapcontaining the requested `:filter` definition (in this example: for the `:filter` named `state_change_only`).

```
curl
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/state_change_only -H
"Authorization: Bearer $SENSU_TOKEN"
{
  "metadata": {
    "name": "state_change_only",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "action": "allow",
  "expressions": [
    "event.check.occurrences == 1"
```

```
  ],
  "runtime_assets": []
}
```

API Specification

/filters/:filter (GET)

description	Returns a filter.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters/state_change_only
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)
output	

```
{
  "metadata": {
    "name": "state_change_only",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "action": "allow",
  "expressions": [
    "event.check.occurrences == 1"
  ],
  "runtime_assets": []
}
```

`/filters/:filter` (PUT)

API Specification

/filters/:filter (PUT)	
description	Create or update a Sensu filter.
example URL	http://hostname:8080/api/core/v2/namespaces/default/filters/development_filter

payload	<pre>{
	  "metadata": {
	  "name": "development_filter",
	  "namespace": "default",
	  "labels": null,
	  "annotations": null
	  },
	  "action": "deny",
	  "expressions": [
	  "event.entity.metadata.namespace == 'production'"
	  ],
	  "runtime_assets": []
	}</pre>

response codes	☐ **Success**: 201 (Created) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/filters/:filter` (DELETE)

The `/filters/:filter` API endpoint provides HTTP DELETE access to delete a filter from Sensu given the filter name.

EXAMPLE

The following example shows a request to delete the filter `production-only`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/filters/production-only

HTTP/1.1 204 No Content
```

API Specification

/filters/:filter (DELETE)	
description	Removes a filter from Sensu given the filter name.
example url	http://hostname:8080/api/core/v2/namespaces/default/filters/production-only
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Handlers API

The `/handlers` API endpoint

`/handlers` (GET)

The `/handlers` API endpoint provides HTTP GET access to handler data.

EXAMPLE

The following example demonstrates a request to the `/handlers` API, resulting ina JSON Array containing handler definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers -H "Authorizatio
$SENSU_TOKEN"
[
  {
    "metadata": {
```

```
      "name": "slack",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXX
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
]
```

API Specification

/handlers (GET)	
description	Returns the list of handlers.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "slack",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "sensu-slack-handler --channel '#monitoring'",
    "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  },
  {
    "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"
```

```
  }
]
```

`/handlers` (POST)

/handlers (POST)	
description	Create a Sensu handler.
example URL	http://hostname:8080/api/core/v2/namespaces/default/handlers
payload	

```
{
  "metadata": {
    "name": "influx-db",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-influxdb-handler -d sensu",
  "env_vars": [

"INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
    "INFLUXDB_USER=sensu",
    "INFLUXDB_PASSWORD=password"
  ],
  "filters": [],
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
}
```

response ☐ **Success**: 200 (OK)

codes

- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/handlers/:handler` API endpoint

`/handlers/:handler` (GET)

The `/handlers/:handler` API endpoint provides HTTP GET access to handler data for specific `:handler` definitions, by handler `name`.

EXAMPLE

In the following example, querying the `/handlers/:handler` API returns a JSON Mapcontaining the requested `:handler` definition (in this example: for the `:handler` named `slack`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/slack -H "Author
Bearer $SENSU_TOKEN"
{
  "metadata": {
    "name": "slack",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-slack-handler --channel '#monitoring'",
  "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXX
  ],
  "filters": [
    "is_incident",
    "not_silenced"
  ],
  "handlers": [],
  "runtime_assets": [],
```

```
  "timeout": 0,
  "type": "pipe"
}
```

API Specification

/handlers/:handler (GET)

description	Returns a handler.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers/slack
response type	Map
response codes	**Success**: 200 (OK) **Missing**: 404 (Not Found) **Error**: 500 (Internal Server Error)
output	

```
{
  "metadata": {
    "name": "slack",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-slack-handler --channel '#monitoring'",
  "env_vars": [

"SLACK_WEBHOOK_URL=https://hooks.slack.com/services/T00000000/
  ],
  "filters": [
    "is_incident",
    "not_silenced"
  ],
  "handlers": [],
  "runtime_assets": [],
```

```
  "timeout": 0,
  "type": "pipe"
}
```

`/handlers/:handler` (PUT)

API Specification

/handlers/:handler (PUT)	
description	Create or update a Sensu handler.
example URL	http://hostname:8080/api/core/v2/namespaces/default/handlers/influx-db
payload	

```
{
  "metadata": {
    "name": "influx-db",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "sensu-influxdb-handler -d sensu",
  "env_vars": [

"INFLUXDB_ADDR=http://influxdb.default.svc.cluster.local:8086",
    "INFLUXDB_USER=sensu",
    "INFLUXDB_PASSWORD=password"
  ],
  "filters": [],
  "handlers": [],
  "runtime_assets": [],
  "timeout": 0,
  "type": "pipe"
```

```
}
```

response codes	☐ **Success**: 201 (Created) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/handlers/:handler` (DELETE)

The `/handlers/:handler` API endpoint provides HTTP DELETE access to delete a handler from Sensu given the handler name.

EXAMPLE

The following example shows a request to delete the handler `slack`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/handlers/slack

HTTP/1.1 204 No Content
```

API Specification

/handlers/:handler (DELETE)	
description	Removes a handler from Sensu given the handler name.
example url	http://hostname:8080/api/core/v2/namespaces/default/handlers/slack
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found)

 Error: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Health API

The `/health` API endpoint

`/health` (GET)

The `/health` API endpoint provides HTTP GET access to health data for your Sensu instance.

EXAMPLE

The following example demonstrates a request to the `/health` API, resulting ina JSON map containing Sensu health data.

```
curl http://127.0.0.1:8080/health

HTTP/1.1 200 OK
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 9882886658148554927,
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ]
}
```

API Specification

/health (GET)

description	Returns health information about the Sensu instance
example url	http://hostname:8080/health
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
output	


```
{
  "Alarms": null,
  "ClusterHealth": [
    {
      "MemberID": 9882886658148554927,
      "Name": "default",
      "Err": "",
      "Healthy": true
    }
  ]
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Hooks API

The `/hooks` API endpoint

`/hooks` (GET)

The `/hooks` API endpoint provides HTTP GET access to hook data.

EXAMPLE

The following example demonstrates a request to the `/hooks` API, resulting ina JSON Array containing hook definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks -H
"Authorization: Bearer $SENSU_TOKEN"
[
  {
    "metadata": {
```

```
      "name": "process-tree",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "ps aux",
    "timeout": 10,
    "stdin": false
  }
]
```

API Specification

/hooks (GET)

description	Returns the list of hooks.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "metadata": {
      "name": "process-tree",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "ps aux",
    "timeout": 10,
    "stdin": false
  },
  {
```

```
    "metadata": {
      "name": "nginx-log",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "tail -n 100 /var/log/nginx/error.log",
    "timeout": 10,
    "stdin": false
  }
]
```

`/hooks` (POST)

/hooks (POST)	
description	Create a Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks

payload	<pre>{
	  "metadata": {
	    "name": "process-tree",
	    "namespace": "default",
	    "labels": null,
	    "annotations": null
	  },
	  "command": "ps aux",
	  "timeout": 10,
	  "stdin": false
	}</pre>

response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request)

□ **Error**: 500 (Internal Server Error)

The `/hooks/:hook` API endpoint

`/hooks/:hook` (GET)

The `/hooks/:hook` API endpoint provides HTTP GET access to hook data for specific `:hook` definitions, by hook `name`.

EXAMPLE

In the following example, querying the `/hooks/:hook` API returns a JSON Mapcontaining the requested `:hook` definition (in this example: for the `:hook` named `process-tree`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/process-tree -H
"Authorization: Bearer $SENSU_TOKEN"
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "ps aux",
  "timeout": 10,
  "stdin": false
}
```

API Specification

/hooks/:hook
(GET)

description	Returns a hook.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

output	{
	"metadata": {
	"name": "process-tree",
	"namespace": "default",
	"labels": null,
	"annotations": null
	},
	"command": "ps aux",
	"timeout": 10,
	"stdin": false
	}

`/hooks/:hook` (PUT)

API Specification

/hooks/:hook (PUT)	
description	Create or update a Sensu hook.
example URL	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree

payload

```
{
  "metadata": {
    "name": "process-tree",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "ps aux",
  "timeout": 10,
  "stdin": false
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

`/hooks/:hook` (DELETE)

The `/hooks/:hook` API endpoint provides HTTP DELETE access to delete a check hook from Sensu given the hook name.

EXAMPLE

The following example shows a request to delete the hook `process-tree`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/hooks/process-tree

HTTP/1.1 204 No Content
```

API Specification

/hooks/:hook (DELETE)	
description	Removes a hook from Sensu given the hook name.
example url	http://hostname:8080/api/core/v2/namespaces/default/hooks/process-tree
response codes	☐ **Success**: 202 (Accepted) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers

License management API

The license management API is available in Sensu Go version 5.2.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest API documentation.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Mutators API

The `/mutators` API endpoint

`/mutators` (GET)

The `/mutators` API endpoint provides HTTP GET access to mutator data.

EXAMPLE

The following example demonstrates a request to the `/mutators` API, resulting ina JSON Array containing mutator definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators -H
"Authorization: Bearer $SENSU_TOKEN"
[
  {
    "metadata": {
```

```
      "name": "example-mutator",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
  }
]
```


API Specification

/mutators (GET)

description	Returns the list of mutators.
example url	http://hostname:8080/api/core/v2/namespaces/default/mutators
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "metadata": {
      "name": "example-mutator",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
```

```
  }
]
```

`/mutators` (POST)

/mutators (POST)	
description	Create a Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators

payload	<pre>{
	  "metadata": {
	  "name": "example-mutator",
	  "namespace": "default",
	  "labels": null,
	  "annotations": null
	  },
	  "command": "example_mutator.go",
	  "timeout": 0,
	  "env_vars": [],
	  "runtime_assets": []
	}</pre>

response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

The `/mutators/:mutator` API endpoint

`/mutators/:mutator` (GET)

The `/mutators/:mutator` API endpoint provides HTTP GET access to mutator data for specific `:mutator` definitions, by mutator `name`.

EXAMPLE

In the following example, querying the `/mutators/:mutator` API returns a JSON Mapcontaining the requested `:mutator` definition (in this example: for the `:mutator` named `example-mutator`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-
mutator -H "Authorization: Bearer $SENSU_TOKEN"
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
```

API Specification

/mutators/:mutator (GET)	
description	Returns a mutator.
example url	http://hostname:8080/api/core/v2/namespaces/default/mutators/mutator-name
response type	Map

response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

output


```
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
```


`/mutators/:mutator` (PUT)

API Specification

/mutators/:mutator (PUT)

description	Create or update a Sensu mutator.
example URL	http://hostname:8080/api/core/v2/namespaces/default/mutators/example-mutator

payload


```
{
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
```

```
    "annotations": null
  },
  "command": "example_mutator.go",
  "timeout": 0,
  "env_vars": [],
  "runtime_assets": []
}
```

---

response codes

- **Success**: 201 (Created)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

`/mutators/:mutator` (DELETE)

The `/mutators/:mutator` API endpoint provides HTTP DELETE access to delete a mutator from Sensu given the mutator name.

EXAMPLE

The following example shows a request to delete the mutator `example-mutator`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/mutators/example-mutator

HTTP/1.1 204 No Content
```

API Specification

/mutators/:mutator (DELETE)

description	Removes a mutator from Sensu given the mutator name.
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)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Namespaces API

The `/namespaces` API endpoint

`/namespaces` (GET)

The `/namespaces` API endpoint provides HTTP GET access to namespace data.

EXAMPLE

The following example demonstrates a request to the `/namespaces` API, resulting ina JSON Array containing namespace definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces -H "Authorization: Bearer
$SENSU_TOKEN"
[
  {
    "name": "default"
  },
```

```
  {
    "name": "development"
  }
]
```

API Specification

/namespaces (GET)	
description	Returns the list of namespaces.
example url	http://hostname:8080/api/core/v2/namespaces
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)

output	<pre>[
	  {
	  "name": "default"
	  },
	  {
	  "name": "development"
	  }
	]</pre>

`/namespaces` (POST)

/namespaces (POST)

description	Create a Sensu namespace.
example URL	http://hostname:8080/api/core/v2/namespaces

payload	{
	"name": "development"
	}

response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

The `/namespaces/:namespace` API endpoint

`/namespaces/:namespace` (PUT)

API Specification

/namespaces/:namespace (PUT)	
description	Create or update a Sensu namespace.
example URL	http://hostname:8080/api/core/v2/namespaces/development

payload	{
	"name": "development"
	}

response codes	☐ **Success**: 201 (Created) ☐ **Malformed**: 400 (Bad Request)

- **Error**: 500 (Internal Server Error)

`/namespaces/:namespace` (DELETE)

The `/namespaces/:namespace` API endpoint provides HTTP DELETE access to delete a namespace from Sensu given the namespace name.

EXAMPLE

The following example shows a request to delete the namespace `development`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/development

HTTP/1.1 204 No Content
```

API Specification

/namespaces/:namespace (DELETE)	
description	Removes a namespace from Sensu given the namespace name.
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)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

API overview

Sensu Go 5.0 includes API v2.

The Sensu backend REST API provides access to Sensu workflow configurations and monitoring event data.For the Sensu agent API, see the agent reference.

URL format

Sensu API endpoints use the standard URL format `/api/{group}/{version}/namespaces/{namespace}` where:

- `{group}` is the API group. All currently existing Sensu API endpoints are of group `core`.
- `{version}` is the API version. Sensu Go 5.0 uses API v2.
- `{namespace}` is the namespace name. The examples in these API docs use the `default` namespace. The Sensu API requires that the authenticated user have the correct access permissions for the namespace specified in the URL. If the authenticated user has the correct cluster-wide permissions, you can leave out the `/namespaces/{namespace}` portion of the URL to access Sensu resources across namespaces. See the RBAC reference for more information about configuring Sensu users and access controls.

Data format

The API uses JSON formatted requests and responses.In terms of sensuctl output types, the Sensu API uses the `json` format, not `wrapped-json`.

Versioning

The Sensu Go API is versioned according to the format `v{majorVersion}{stabilityLevel}{iterationNumber}` `v2`

, in which is stable version 2.The Sensu API guarantees backward compatibility for stable versions of the API.

Sensu makes no guarantee that an alpha or beta API will be maintained for any period of time.Alpha versions should be considered under active development and may not be published for every release.Beta APIs, while more stable than alpha versions, offer similarly short-lived lifespans and also provide no guarantee of programmatic conversions when the API is updated.

Access control

With the exception of the health API, the Sensu API requires authentication using a JWT access token. Sensuctl provides an easy way to generate access tokens for short-lived use with the Sensu API.The user credentials that you use to log in to sensuctl determine your permissions to get, list, create, update, and delete resources using the Sensu API.

To generate an API access token using sensuctl:

1. Install and log in to sensuctl.
2. Retrieve an access token for your user:

   ```
   cat ~/.config/sensu/sensuctl/cluster|grep access_token
   ```

   The access token should be included in the output:

   ```
   "access_token": "eyJhbGciOiJIUzI1NiIs...",
   ```

3. Copy the access token into the authentication header of the API request. For example:

   ```
   curl http://127.0.0.1:8080/api/core/v2/namespaces/default/events -H
   "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..."
   ```

Access tokens last for around 15 minutes.If your token expires, you should see a 401 Unauthorized response from the API.

To create a new token, first run any sensuctl command (like `sensuctl event list`) then repeat the steps above.

Request size

API request bodies are limited to 0.512 MB in size.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Role bindings API

The `/rolebindings` API endpoint

`/rolebindings` (GET)

The `/rolebindings` API endpoint provides HTTP GET access to role binding data.

EXAMPLE

The following example demonstrates a request to the `/rolebindings` API, resulting ina JSON Array containing role binding definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings -H
"Authorization: Bearer $SENSU_TOKEN"

HTTP/1.1 200 OK
[
```

```
  {
    "subjects": [
      {
        "type": "Group",
        "name": "readers"
      }
    ],
    "role_ref": {
      "type": "Role",
      "name": "read-only"
    },
    "metadata": {
      "name": "readers-group-binding",
      "namespace": "default"
    }
  }
]
```

API Specification

/rolebindings (GET)	
description	Returns the list of role bindings.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)
output	

```
[
  {
    "subjects": [
      {
        "type": "Group",
```

```
        "name": "readers"
      }
    ],
    "role_ref": {
      "type": "Role",
      "name": "read-only"
    },
    "metadata": {
      "name": "readers-group-binding",
      "namespace": "default"
    }
  }
]
```

`/rolebindings` (POST)

/rolebindings (POST)	
description	Create 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**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/rolebindings/:rolebinding` API endpoint

`/rolebindings/:rolebinding` (GET)

The `/rolebindings/:rolebinding` API endpoint provides HTTP GET access to role binding data for specific `:rolebinding` definitions, by role binding `name`.

EXAMPLE

In the following example, querying the `/rolebindings/:rolebinding` API returns a JSON Map containing the requested `:rolebinding` definition (in this example: for the `:rolebinding` named `readers-group-binding`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/readers-group-binding -H "Authorization: Bearer $SENSU_TOKEN"

HTTP/1.1 200 OK
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
```

```
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
    "namespace": "default"
  }
}
```

API Specification

/rolebindings/:rolebinding (GET)

description	Returns a role binding.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/readers-group-binding
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)
output	

```
{
  "subjects": [
    {
      "type": "Group",
      "name": "readers"
    }
  ],
  "role_ref": {
    "type": "Role",
    "name": "read-only"
  },
  "metadata": {
    "name": "readers-group-binding",
```

```
    "namespace": "default"
  }
}
```

`/rolebindings/:rolebinding` (PUT)

API Specification

/rolebindings/:rolebinding (PUT)	
description	Create or update a Sensu role binding.
example URL	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/group-binding

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)

`/rolebindings/:rolebinding` (DELETE)

The `/rolebindings/:rolebinding` API endpoint provides HTTP DELETE access to delete a role binding from Sensu given the role binding name.

EXAMPLE

The following example shows a request to delete the role binding `dev-binding`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/rolebindings/dev-binding

HTTP/1.1 204 No Content
```

API Specification

/rolebindings/:rolebinding (DELETE)	
description	Removes a role binding from Sensu given the role binding name.
example url	http://hostname:8080/api/core/v2/namespaces/default/rolebindings/binding
response codes	**Success**: 202 (Accepted) **Missing**: 404 (Not Found) **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Roles API

The `/roles` API endpoint

`/roles` (GET)

The `/roles` API endpoint provides HTTP GET access to role data.

EXAMPLE

The following example demonstrates a request to the `/roles` API, resulting ina JSON Array containing role definitions.

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/roles -H
"Authorization: Bearer $SENSU_TOKEN"

HTTP/1.1 200 OK
[
```

```
  {
    "rules": [
      {
        "verbs": [
          "read"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "read-only",
      "namespace": "default"
    }
  }
]
```


API Specification

/roles (GET)	
description	Returns the list of roles.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)
output	


```
[
  {
    "rules": [
      {
        "verbs": [
```

```
          "read"
        ],
        "resources": [
          "*"
        ],
        "resource_names": null
      }
    ],
    "metadata": {
      "name": "read-only",
      "namespace": "default"
    }
  }
]
```

`/roles` (POST)

/roles (POST)	
description	Create a Sensu role.
example URL	http://hostname:8080/api/core/v2/namespaces/default/roles
payload	

```
{
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": []
    }
  ],
```

```
  "metadata": {
    "name": "event-reader",
    "namespace": "default"
  }
}
```

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/roles/:role` API endpoint

`/roles/:role` (GET)

The `/roles/:role` API endpoint provides HTTP GET access to role data for specific `:role` definitions, by role `name`.

EXAMPLE

In the following example, querying the `/roles/:role` API returns a JSON Mapcontaining the requested `:role` definition (in this example: for the `:role` named `read-only`).

```
curl http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/read-only -H
"Authorization: Bearer $SENSU_TOKEN"

HTTP/1.1 200 OK
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
```

```
        "*"
      ],
      "resource_names": null
    }
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}
```

API Specification

/roles/:role (GET)	
description	Returns a role.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles/read-only
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)
output	

```
{
  "rules": [
    {
      "verbs": [
        "read"
      ],
      "resources": [
        "*"
      ],
      "resource_names": null
    }
```

```
  ],
  "metadata": {
    "name": "read-only",
    "namespace": "default"
  }
}
```

`/roles/:role` (PUT)

API Specification

/roles/:role (PUT)	
description	Create or update a Sensu role.
example URL	http://hostname:8080/api/core/v2/namespaces/default/roles/event-reader
payload	

```
{
  "rules": [
    {
      "verbs": [
        "get",
        "list"
      ],
      "resources": [
        "events"
      ],
      "resource_names": []
    }
  ],
  "metadata": {
    "name": "event-reader",
    "namespace": "default"
```

```
    }
  }
```

response codes	☐ **Success**: 201 (Created) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/roles/:role` (DELETE)

The `/roles/:role` API endpoint provides HTTP DELETE access to delete a role from Sensu given the role name.

EXAMPLE

The following example shows a request to delete the role `read-only`, resulting in a successful HTTP 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/roles/ready-only

HTTP/1.1 204 No Content
```

API Specification

/roles/:role (DELETE)	
description	Removes a role from Sensu given the role name.
example url	http://hostname:8080/api/core/v2/namespaces/default/roles/ready-only

response codes

- **Success**: 204 (No Content)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Silencing API

The `/silenced` API endpoint

`/silenced` (GET)

The `/silenced` API endpoint provides HTTP GET access to silencing entry data.

EXAMPLE

The following example demonstrates a request to the `/silenced` API, resulting ina JSON Array containing silencing entry definitions.

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```

API Specification

/silenced (GET)

description	Returns the list of silencing entries.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced
response type	Array
response codes	**Success**: 200 (OK) **Error**: 500 (Internal Server Error)
output	`[ { "metadata": {`

```
      "name": "linux:check-cpu",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```


`/silenced` (POST)

/silenced (POST)	
description	Create a Sensu silencing entry.
example URL	http://hostname:8080/api/core/v2/namespaces/default/silenced
payload	`{ "metadata": { "name": "linux:check-cpu", "namespace": "default", "labels": null, "annotations": null }, "expire": -1, "expire_on_resolve": false, "creator": "admin", "subscription": "linux", "begin": 1542671205 }`

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/silenced/:silenced` API endpoint

`/silenced/:silenced` (GET)

The `/silenced/:silenced` API endpoint provides HTTP GET access to silencing entry data for specific `:silenced` definitions, by silencing entry `name`.

EXAMPLE

In the following example, querying the `/silenced/:silenced` API returns a JSON Mapcontaining 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 -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu

HTTP/1.1 200 OK
{
  "metadata": {
    "name": "linux:check-cpu",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "expire": -1,
  "expire_on_resolve": false,
  "creator": "admin",
  "subscription": "linux",
  "begin": 1542671205
```

```
}
```

API Specification

/silenced/:silenced (GET)	
description	Returns a silencing entry.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu
response type	Map
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

output	{
	"metadata": {
	"name": "linux:check-cpu",
	"namespace": "default",
	"labels": null,
	"annotations": null
	},
	"expire": -1,
	"expire_on_resolve": false,
	"creator": "admin",
	"subscription": "linux",
	"begin": 1542671205
	}

`/silenced/:silenced` (PUT)

API Specification

/silenced/:silenced (PUT)	
description	Create or update a Sensu silencing entry.
example URL	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu

payload	{
	"metadata": {
	"name": "linux:check-cpu",
	"namespace": "default",
	"labels": null,
	"annotations": null
	},
	"expire": -1,
	"expire_on_resolve": false,
	"creator": "admin",
	"subscription": "linux",
	"begin": 1542671205
	}

response codes	**Success**: 201 (Created) **Malformed**: 400 (Bad Request) **Error**: 500 (Internal Server Error)

`/silenced/:silenced` (DELETE)

The `/silenced/:silenced` API endpoint provides HTTP DELETE access to delete a silencing entry by silencing entry `name`.

EXAMPLE

In the following example, querying the `/silenced/:silenced` API to delete the the silencing entry named `linux:check-cpu` results in a successful 204 No Content response.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu

HTTP/1.1 204 No Content
```

API Specification

/silenced/:silenced (DELETE)	
description	Removes a silencing entry from Sensu given the silencing entry name.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/linux:check-cpu
response codes	☐ **Success**: 204 (No Content) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

The `/silenced/subscriptions/:subscription` API endpoint

`/silenced/subscriptions/:subscription` (GET)

The `/silenced/subscriptions/:subscription` API endpoint provides HTTP GET access to silencing entry data by subscription `name`.

EXAMPLE

In the following example, querying the `silenced/subscriptions/:subscription` API returns a JSON Arraycontaining the requested silencing entries for the given subscription (in this example: for the `linux` subscription).

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/subscriptions/linux

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```

API Specification

/silenced/subscriptions/:subscription (GET)	
description	Returns all silencing entries for the specified subscription.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/subscriptions/linu
response type	Array

response codes

- **Success**: 200 (OK)
- **Missing**: 404 (Not Found)
- **Error**: 500 (Internal Server Error)

---

output


```
[
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "subscription": "linux",
    "begin": 1542671205
  }
]
```


The `/silenced/checks/:check` API endpoint

`/silenced/checks/:check` (GET)

The `/silenced/checks/:check` API endpoint provides HTTP GET access to silencing entry data by check `name`.

EXAMPLE

In the following example, querying the `silenced/checks/:check` API returns a JSON Arraycontaining the requested silencing entries for the given check (in this example: for the `check-cpu` check).

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/namespaces/default/silenced/checks/check-cpu

HTTP/1.1 200 OK
[
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "check": "linux",
    "begin": 1542671205
  }
]
```

API Specification

/silenced/checks/ :check (GET)	
description	Returns all silencing entries for the specified check.
example url	http://hostname:8080/api/core/v2/namespaces/default/silenced/checks/chec cpu
response type	Array
response codes	☐ **Success**: 200 (OK) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)
output	`[`

```
  {
    "metadata": {
      "name": "linux:check-cpu",
      "namespace": "default",
      "labels": null,
      "annotations": null
    },
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "check": "linux",
    "begin": 1542671205
  }
]
```


About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Tessen API

The Tessen API is available in Sensu Go version 5.5.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest API documentation.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Users API

The `/users` API endpoint

`/users` (GET)

The `/users` API endpoint provides HTTP GET access to user data.

EXAMPLE

The following example demonstrates a request to the `/users` API, resulting ina JSON Array containing user definitions.

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users

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

response type	Array

response codes	☐ **Success**: 200 (OK) ☐ **Error**: 500 (Internal Server Error)

output


```
[
  {
    "username": "admin",
    "groups": [
      "cluster-admins"
    ],
    "disabled": false
  },
  {
    "username": "agent",
    "groups": [
      "system:agents"
    ],
    "disabled": false
  }
]
```


`/users` (POST)

The `/users` API endpoint provides HTTP POST access to create a user.

EXAMPLE

The following example demonstrates a POST request to the `/users` API to create the user `alice`, resulting in an HTTP 200 response and the created user definition.

```
curl -X POST \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
```

```
-d '{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "temporary",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users

HTTP/1.1 200 OK
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

API Specification

/users (POST)	
description	Create a Sensu user.
example URL	http://hostname:8080/api/core/v2/users

payload	{
	  "username": "alice",
	  "groups": [
	  "ops"
	  ],
	  "password": "temporary",
	  "disabled": false
	}

payload parameters

- `username` (string, required)
- `password` (string, required): Must have at least eight characters
- `groups` (array): Sets of shared permissions applicable to this user
- `disabled`: When set to `true`, invalidates user credentials and permissions

---

response codes

- **Success**: 200 (OK)
- **Malformed**: 400 (Bad Request)
- **Error**: 500 (Internal Server Error)

The `/users/:user` API endpoint

`/users/:user` (GET)

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 Mapcontaining the requested `:user` definition (in this example: for the `alice` user).

```
curl -H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice

HTTP/1.1 200 OK
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

API Specification

/users/:user (GET)	
description	Returns a user given the username as a URL parameter.
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	{
	"username": "alice",
	"groups": [
	"ops"
	],
	"disabled": **false**
	}

`/users/:user` (PUT)

EXAMPLE

The following example demonstrates a PUT request to the `/users` API to update the user `alice`, in this case to reset their password, resulting in an HTTP 200 response and the updated user definition.

```
curl -X PUT \
```

```
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "password": "reset-password",
  "disabled": false
}' \
http://127.0.0.1:8080/api/core/v2/users/alice

HTTP/1.1 200 OK
{
  "username": "alice",
  "groups": [
    "ops"
  ],
  "disabled": false
}
```

API Specification

/users/:user (PUT)	
description	Create or update a Sensu user given the username.
example URL	http://hostname:8080/api/core/v2/users/alice

payload	{
	  "username": "alice",
	  "groups": [
	  "ops"
	  ],
	  "password": "reset-password",
	  "disabled": false
	}

response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/users/:user` (DELETE)

EXAMPLE

In the following example, an HTTP DELETE request is submitted to the `/users/:user` API to disable the user `alice`, resulting in a successful 204 (No Content) HTTP response code.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice

HTTP/1.1 204 No Content
```

API Specification

/users/:user (DELETE)	
description	Disables a user given the username as a URL parameter.
example url	http://hostname:8080/api/core/v2/users/alice
response codes	☐ **Success**: 204 (No Content) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

The `/users/:user/password` API endpoint

`/users/:user/password` (PUT)

The `/users/:user/password` API endpoint provides HTTP PUT access to update a user's password.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/users/:user/password` API to update the password for the user `alice`, resulting in a 200 (OK) HTTP response code.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
-d '{
  "username": "alice",
  "password": "newpassword"
}' \
http://127.0.0.1:8080/api/core/v2/users/alice/password

HTTP/1.1 200 OK
```

API Specification

/users/:user/password (PUT)	
description	Update the password for a Sensu user.
example URL	http://hostname:8080/api/core/v2/users/alice/password
payload	`{ "username": "admin", "password": "newpassword" }`

```
}
```

payload parameters	☐ `username` (string, required): the `username` for the Sensu user ☐ `password` (string, required): the user's new password
response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

The `/users/:user/reinstate` API endpoint

`/users/:user/reinstate` (PUT)

The `/users/:user/reinstate` API endpoint provides HTTP PUT access to re-enable a disabled user.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/users/:user/reinstate` API to enable the disabled user `alice`, resulting in a 200 (OK) HTTP response code.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
-H 'Content-Type: application/json' \
http://127.0.0.1:8080/api/core/v2/users/alice/reinstate

HTTP/1.1 200 OK
```

API Specification

/users/:user/reinstate (PUT)	
description	Reinstate a disabled user.
example URL	http://hostname:8080/api/core/v2/users/alice/reinstate
response codes	☐ **Success**: 200 (OK) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

The `/users/:user/groups` API endpoint

`/users/:user/groups` (DELETE)

The `/users/:user/groups` API endpoint provides HTTP DELETE access to remove a user from all groups.

EXAMPLE

In the following example, an HTTP DELETE request is submitted to the `/users/:user/groups` API to remove the user `alice` from all groups within Sensu, resulting in a successful 204 (No Content) HTTP response code.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups

HTTP/1.1 204 No Content
```

API Specification

/users/:user/groups

(DELETE)	
description	Removes a user from all groups.
example url	http://hostname:8080/api/core/v2/users/alice/groups
response codes	☐ **Success**: 204 (No Content) ☐ **Missing**: 404 (Not Found) ☐ **Error**: 500 (Internal Server Error)

The `/users/:user/groups/:group` API endpoints

`/users/:user/groups/:group` (PUT)

The `/users/:user/groups/:group` API endpoint provides HTTP PUT access to assign a user to a group.

EXAMPLE

In the following example, an HTTP PUT request is submitted to the `/users/:user/groups/:group` API to add the user `alice` to the group `ops`, resulting in a successful 204 (No Content) HTTP response code.

```
curl -X PUT \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops

HTTP/1.1 204 No Content
```

API Specification

/users/:user/groups/:group (PUT)

description	Add a user to a group.
example URL	http://hostname:8080/api/core/v2/users/alice/groups/ops
payload	
response codes	☐ **Success**: 204 (No Content) ☐ **Malformed**: 400 (Bad Request) ☐ **Error**: 500 (Internal Server Error)

`/users/:user/groups/:group` (DELETE)

The `/users/:user/groups/:group` API endpoint provides HTTP DELETE access to remove a user from a group.

EXAMPLE

In the following example, an HTTP DELETE request is submitted to the `/users/:user/groups/:group` API to remove the user `alice` from the group `ops`, resulting in a successful 204 (No Content) HTTP response code.

```
curl -X DELETE \
-H "Authorization: Bearer $SENSU_TOKEN" \
http://127.0.0.1:8080/api/core/v2/users/alice/groups/ops

HTTP/1.1 204 No Content
```

API Specification

/users/:user/groups/:group (DELETE)

description	Removes a user from a 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)

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

sensuctl CLI

- Quickstart
- Reference

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Sensuctl quick reference

Quick reference

```
# Configure and log in with defaults
sensuctl configure
? Sensu Backend URL: http://127.0.0.1:8080
? Username: admin
? Password: P@ssw0rd!

# Create resources from a file containing JSON resource definitions
sensuctl create --file filename.json

# See monitored entities
sensuctl entity list

# See monitoring events
sensuctl event list

# Edit a check named check-cpu
sensuctl edit check check-cpu

# See the JSON configuration for a check named check-cpu
sensuctl check info check-cpu --format wrapped-json
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Sensuctl

Sensuctl is a command line tool for managing resources within Sensu. It works bycalling Sensu's underlying API to create, read, update, and delete resources,events, and entities. Sensuctl is available for Linux, macOS, and Windows.See the installation guide to install and configure sensuctl.

Getting help

Sensuctl supports a `--help` flag for each command and subcommand.

```
# See command and global flags
sensuctl --help

# See subcommands and flags
sensuctl check --help

# See usage and flags
sensuctl check delete --help
```

First-time setup

To set up sensuctl, run `sensuctl configure` to log in to sensuctl and connect to the Sensu backend.

```
sensuctl configure
```

When prompted, input the Sensu backend URL and your Sensu access credentials.

```
? Sensu Backend URL: http://127.0.0.1:8080
? Username: admin
? Password: P@ssw0rd!
? Namespace: default
? Preferred output format: tabular
```

Sensu backend URL

The HTTP or HTTPS URL where sensuctl can connect to the Sensu backend server, defaulting to `http://127.0.0.1:8080`.When connecting to a Sensu cluster, connect sensuctl to any single backend in the cluster.For more information on configuring the Sensu backend URL, see the backend reference.

Username | password | namespace

By default, Sensu includes a user named `admin` with password `P@ssw0rd!` 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, see the RBAC reference.

Preferred output format

Sensuctl supports the following output formats:

- `tabular`: user-friendly, columnar format
- `wrapped-json`: accepted format for use with `sensuctl create`
- `yaml`: accepted format for use with `sensuctl create`
- `json`: format used by the Sensu API

Once logged in, you can change the output format using `sensuctl config set-format` or set it per command using the `--format` flag.

Non-interactive

You can run `sensuctl configure` non-interactively using the `-n` (`--non-interactive`) flag.

```
sensuctl configure -n --url http://127.0.0.1:8080 --username admin --password
P@ssw0rd! --format tabular
```

Managing 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
```

Sensuctl configuration includes the Sensu backend url, Sensu edition (Core or Enterprise), the default output format for the current user, and the default namespace for the current user.

```
api-url: http://127.0.0.1:8080
```

```
edition: core
format: wrapped-json
namespace: default
```

Set output format

You can 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

You can use the `set-namespace` command to change the default namespace for the current user. For more information about configuring Sensu access control, see the RBAC reference .For example, to change the default namespace to `development`:

```
sensuctl config set-namespace development
```

Log out of sensuctl

To log out of sensuctl:

```
sensuctl logout
```

To log back in:

```
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
--namespace string       namespace in which we perform actions (default:
"default")
```

Creating resources

The `sensuctl create` command allows you to create or update resources by reading from STDIN or a flag configured file (`-f`).The `create` command accepts Sensu resource definitions in `wrapped-json` and `yaml` .Both JSON and YAML resource definitions wrap the contents of the resource in `spec` and identify the resource `type` (see below for an example, and this table for a list of supported types).See the reference docs for information about creating resource definitions.

For example, the following file `my-resources.json` specifies two resources: a `marketing-site` check and a `slack` handler.

```
{
  "type": "CheckConfig",
  "spec": {
    "command": "check-http.go -u https://dean-learner.book",
    "subscriptions": ["demo"],
    "interval": 15,
```

```
    "handlers": ["slack"],
    "metadata" : {
      "name": "marketing-site",
      "namespace": "default"
    }
  }
}
{
  "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
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
}
```

NOTE: Commas cannot be included between JSON resource definitions when using `sensuctl create`.

To create all resources from `my-resources.json` using `sensuctl create`:

```
sensuctl create --file my-resources.json
```

Or:

```
cat my-resources.json | sensuctl create
```

sensuctl create resource types

sensuctl create types			
`AdhocRequest`	`adhoc_request`	`Asset`	`asset`
`CheckConfig`	`check_config`	`ClusterRole`	`cluster_role`
`ClusterRoleBinding`	`cluster_role_binding`	`Entity`	`entity`
`Event`	`event`	`EventFilter`	`event_filter`
`Handler`	`handler`	`Hook`	`hook`
`HookConfig`	`hook_config`	`Mutator`	`mutator`
`Namespace`	`namespace`	`Role`	`role`
`RoleBinding`	`role_binding`	`Silenced`	`silenced`

Updating resources

Sensuctl allows you to update resource definitions using a text editor.To use `sensuctl edit`, specify the resource type and resource name.

For example, to edit a handler named `slack` using `sensuctl edit`:

```
sensuctl edit handler slack
```

sensuctl edit resource types

sensuctl edit types			
`asset`	`check`	`cluster`	`cluster-role`
`cluster-role-binding`	`entity`	`event`	`filter`
`handler`	`hook`	`mutator`	`namespace`
`role`	`role-binding`	`silenced`	`user`

Managing resources

Sensuctl provides the following commands to manage Sensu resources.

- `sensuctl asset`
- `sensuctl check`
- `sensuctl cluster`
- `sensuctl cluster-role`
- `sensuctl cluster-role-binding`
- `sensuctl entity`
- `sensuctl event`
- `sensuctl filter`
- `sensuctl handler`
- `sensuctl hook`
- `sensuctl mutator`
- `sensuctl namespace`
- `sensuctl role`

- `sensuctl role-binding`
- `sensuctl silenced`
- `sensuctl user`

Subcommands

Sensuctl provides a standard set of list, info, and delete operations for most resource types.

```
list                        list resources
info NAME                   show detailed resource information given resource
name
delete NAME                 delete resource given resource name
```

For example, to list all monitoring checks:

```
sensuctl check list
```

To list checks from all namespaces:

```
sensuctl check list --all-namespaces
```

To write all checks to `my-resources.json` in `wrapped-json` format:

```
sensuctl check list --format wrapped-json > my-resources.json
```

To see the definition for a check named `check-cpu` in `wrapped-json` format:

```
sensuctl check info check-cpu --format wrapped-json
```

In addition to the standard operations, commands may support subcommands or flags that allow you to take special action based on the resource type; the following sections call out those resource-

specific operations.For a list of subcommands specific to a resource, run `sensuctl TYPE --help`.

sensuctl check

In addition to the standard subcommands, sensuctl provides a command to execute 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
member-add       add cluster member to an existing cluster, with comma-separated
peer addresses
member-list      list cluster members
member-remove    remove cluster member by ID
member-update    update cluster member by ID with comma-separated peer addresses
```

To view cluster members:

```
sensuctl cluster member-list
```

To see the health of your Sensu cluster:

```
sensuctl cluster health
```

sensuctl event

In addition to the standard subcommands, sensuctl provides a command to resolve an event.

```
sensuctl event resolve ENTITY CHECK
```

For example, the following command manually resolves an event created by the entity `webserver1` and the check `check-http`:

```
sensuctl event resolve webserver1 check-http
```

sensuctl namespace

See the RBAC reference for information about using access control with namespaces.

sensuctl user

See the RBAC reference for information about local user management with sensuctl.

Time formats

Sensuctl supports multiple time formats depending on the manipulated resource.Supported canonical time zone IDs are defined in the tz database.

WARNING: Canonical zone IDs (i.e. `America/Vancouver`) are not supported onWindows.

Dates with time

Full dates with time are used to specify an exact point in time, which can beused with silencing entries, for example. The following formats are supported:

- RFC3339 with numeric zone offset: `2018-05-10T07:04:00-08:00` or `2018-05-10T15:04:00Z`
- RFC3339 with space delimiters and numeric zone offset: `2018-05-10 07:04:00␣08:00`
- Sensu alpha legacy format with canonical zone ID: `May 10 2018 7:04AM␣America/Vancouver`

Shell auto-completion

Installation (Bash Shell)

Make sure bash completion is installed. If you use a current Linuxin a non-minimal installation, bash completion should be available.On macOS, install with:

```
brew install bash-completion
```

Then add the following to your `~/.bash_profile`:

```
if [ -f $(brew --prefix)/etc/bash_completion ]; then
. $(brew --prefix)/etc/bash_completion
fi
```

Once bash-completion is available, add the following to your `~/.bash_profile`:

```
source <(sensuctl completion bash)
```

You can now source your `~/.bash_profile` or launch a new terminal to utilize completion.

```
source ~/.bash_profile
```

Installation (ZSH)

Add the following to your `~/.zshrc`:

```
source <(sensuctl completion zsh)
```

You can now source your `~/.zshrc` or launch a new terminal to utilize completion.

```
source ~/.zshrc
```

Usage

`sensuctl` Tab

```
check       configure   event       user
asset       completion  entity      handler
```

`sensuctl check` Tab

```
create  delete  import  list
```

Configuration files

During configuration, sensuctl creates configuration files that contain information for connecting to

your Sensu Go deployment. You can find them at `$HOME/.config/sensu/sensuctl/profile` and `$HOME/.config/sensu/sensuctl/profile`. For example:

```
cat .config/sensu/sensuctl/profile
{
  "format": "tabular",
  "namespace": "demo"
}
```

```
cat .config/sensu/sensuctl/cluster
{
  "api-url": "http://localhost:8080",
  "trusted-ca-file": "",
  "insecure-skip-tls-verify": false,
  "access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "expires_at": 1550082282,
  "refresh_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

These are useful if you want to know what cluster you're connecting to, or what namespace you're currently configured to use.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com,

and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Reference

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Sensu agent

All Platforms

The Sensu agent is a lightweight client that runs on the infrastructure components you want to monitor.Agents register with the Sensu backend as monitoring entities with `type: "agent"`.Agent entities are responsible for creating check and metrics events to send to the backend event pipeline.

The Sensu agent is available for Linux, macOS, and Windows.See the installation guide to install the agent.

Creating monitoring events using service checks

Sensu's use of the publish/subscribe pattern of communication allows for automated registration and deregistration of ephemeral systems.At the core of this model are Sensu agent subscriptions.

Each Sensu agent has a defined set of `subscriptions`, a list of roles and responsibilities assigned to the system (for example: a webserver or database).These subscriptions determine which monitoring checks are executed by the agent.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 order 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 matching attribute values in the check definition.
2. Fetches assets and stores them in its local cache. By default, agents cache asset data at `/var/cache/sensu/sensu-agent` (`C:\\ProgramData\sensu\cache\sensu-agent` on Windows systems) or as specified by the the `cache-dir` flag.
3. Executes the check `command`.
4. Executes any hooks specified by the check based on the exit status.
5. Creates an event containing information about the applicable entity, check, and metric.

Subscription configuration

To configure subscriptions for an agent, set the `subscriptions` flag.To configure subscriptions for a check, set the check definition attribute `subscriptions`.

In addition to the subscriptions defined in the agent configuration, Sensu agent entities also subscribe automatically to a subscription matching their entity `name`.For example, an agent entity with the `name: "i-424242"` subscribes to check requests with the subscription `entity:i-424242`.This makes it possible to generate ad-hoc check requests targeting specific entities via the API.

Proxy entities

Sensu proxy entities allow Sensu to monitor external resources on systems or devices where a Sensu agent cannot be installed (such a network switch).Unlike agent entities, proxy entity definitions are stored by the Sensu backend.When the backend requests a check that includes a `proxy_entity_name`, the agent includes the provided entity information in the event data in place of the agent entity data.See the entity reference and the guide to monitoring external resources for more information about monitoring proxy entities.

Creating monitoring events using the agent API

The Sensu agent API allows external sources to send monitoring data to Sensu without needing 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 a 404 Not Found response.

`/events` (POST)

The `/events` API provides HTTP POST access to publish monitoring events to the Sensu backend pipeline via the agent API.

Example

In the following example, an HTTP POST is submitted to the `/events` API, creating an event for a check named `check-mysql-status` with the output `could not connect to mysql` and a status of `1` (warning), resulting in a 201 (Created) HTTP response code.

```
curl -X POST \
-H 'Content-Type: application/json' \
-d '{
  "check": {
    "metadata": {
      "name": "check-mysql-status"
    },
    "status": 1,
    "output": "could not connect to mysql"
  }
}' \
```

```
http://127.0.0.1:3031/events

HTTP/1.1 201 Created
```

PRO TIP: You can use the agent API `/events` endpoint to create proxy entities by including a `proxy_entity_name` attribute within the `check` scope.

Detecting silent failures

You can use the Sensu agent API in combination with the check time-to-live attribute (TTL) to detect silent failures, creating what's commonly referred to as a "dead man's switch" (source: Wikipedia).By using check TTLs, Sensu is able to 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 on check TTLs, see the the check reference.

A great use case for the Sensu agent API is to enable tasks which run outside of Sensu's check scheduling to emit events. Using the check TTL attribute, these events create a dead man's switch, ensuring that if the task fails for any reason, the lack of an "all clear" event from the task notifies 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.

The following is an example of external event input via the Sensu agent API using a check TTL to create a dead man's switch for MySQL backups.If we assume that a MySQL backup script runs periodically and that we expect the job to take a little less than 7 hours to complete, in the case where the job completes successfully, we'd like a record of it but don't need to be alerted. If the job fails for some reason, or continues running past the expected 7 hours, we'd like to be alerted. In the following example, the script sends an event which 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, we have recorded in the Sensu backend that our script started, and we've configured the dead man's switch so that we'll be alerted if the job fails or runs too long. Although it is possible for our script to handle errors gracefully and emit additional monitoring events, this approach allows us to worry less about handling every possible error case, as the lack of additional events before the 7 hour period elapses results in an alert.

If our backup script runs successfully, we 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
```

By omitting the TTL attribute from this event, the dead man's switch being monitored by the Sensu backend is also removed, effectively sounding 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	☐ `check` (required): All check data must be within the `check` scope. ☐ `metadata` (required): The `check` scope must contain a `metadata` scope. ☐ `name` (required): The `metadata` scope must contain the `name` attribute with a string representing the name of the monitoring check. ☐ Any other attributes supported by the Sensu check specification (optional)
response codes	☐ **Success**: 201 (Created) ☐ **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 is submitted to the `/healthz` API:

```
curl http://127.0.0.1:3031/healthz
```

Resulting in a healthy response:

```
ok
```

API specification

/healthz (GET)	
description	Returns `ok` if the agent is active and connected to a Sensu backend; returns `sensu backend unavailable` if the agent is unable to connect to a backend.
example url	http://hostname:3031/healthz

Creating monitoring events using the StatsD listener

Sensu agents include a listener to send StatsD metrics to the event pipeline.By default, Sensu agents listen on UDP socket 8125 (TCP on Windows systems) for messages that follow the StatsD line protocol and send metric events for handling by the Sensu backend.

For example, you can use the Netcat utility to send metrics to the StatsD listener:

```
echo 'abc.def.g:10|c' | nc -w1 -u localhost 8125
```

Metrics received through the StatsD listener are not stored by Sensu, soit's important to configure event handlers.

StatsD line protocol

The Sensu StatsD listener accepts messages formatted according to the StatsD line protocol:

```
<metricname>:<value>|<type>
```

For more information, see the StatsD documentation.

Configuring the StatsD listener

To configure the StatsD listener, specify the `statsd-event-handlers` configuration flag in the agent configuration, and start the agent.

```
# Start an agent that sends StatsD metrics to InfluxDB
sensu-agent --statsd-event-handlers influx-db
```

You can use the StatsD configuration flags to change the default settings for the StatsD listener address, port, and flush interval.

```
# Start an agent with a customized address and flush interval
sensu-agent --statsd-event-handlers influx-db --statsd-flush-interval 1 --statsd-metrics-host 123.4.5.6 --statsd-metrics-port 8125
```

Creating monitoring events using the agent TCP and UDP sockets

NOTE: The agent TCP and UDP sockets are deprecated in favor of the agent events API.

Sensu agents listen for external monitoring data using TCP and UDP sockets.The agent sockets accept JSON event data and pass the event 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.

Using the TCP socket

The following is an example demonstrating 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
```

Using the UDP socket

The following is an example demonstrating 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 backwards compatibility with Sensu 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",
  "client": "sensu-docs-site",
  "executed": 1550013435,
  "duration": 1.903135228
}
```

Socket event specification

The Sensu agent socket ignores any attributes not included in this specification.

name	
description	The check name
required	true
type	String
example	`"name": "check-mysql-status"`

status

description	The 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`, or `2` indicate an `UNKNOWN` or custom status.
required	true
type	Integer
example	`"status": 0`

output

description	The output produced by the check `command`.
required	true
type	String
example	`"output": "CheckHttp OK: 200, 78572 bytes"`

client

description	The name of the Sensu entity associated with the event. The `client` attribute gives you the ability to tie the event to a proxy entity while providing compatibility with Sensu 1.x check results. Use this attribute to specify the name of the proxy entity tied to the event.
required	false
default	The agent entity receiving the event data

type	String
example	`"client": "sensu-docs-site"`

executed

description	The time the check was executed, in seconds since the Unix epoch.
required	false
default	The time the event was received by the agent
type	Integer
example	`"executed": 1458934742`

duration

description	The amount of time (in seconds) it took to execute the check.
required	false
type	Float
example	`"duration": 1.903135228`

command

description	The command executed to produce the event. You can use this attribute to add context to the event data; Sensu does not execute the

	command included in this attribute.
required	false
type	String
example	`"command": "check-http.rb -u https://sensuapp.org"`

interval	
description	The interval used to produce the event. You can use this attribute to add context to the event data; Sensu does not act on the value provided in this attribute.
required	false
default	`1`
type	Integer
example	`"interval": 60`

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-timeout` flag, the Sensu backend creates a keepalive alert in the Sensu dashboard.You can use keepalives to identify unhealthy systems and network partitions, send notifications, trigger auto-remediation, and other useful actions.

NOTE: Keepalive monitoring is not supported for proxy entities, as they are inherently

unable to run a Sensu agent.

Handling keepalive events

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.

Let's say 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:

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "keepalive",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack"
    ]
  }
}
```

Operation

Starting the service

Use the `sensu-agent` tool to start the agent and apply configuration flags.

To start the agent with configuration flags:

```
sensu-agent start --subscriptions disk-checks --log-level debug
```

To see available configuration flags and defaults:

```
sensu-agent start --help
```

If no configuration flags are provided, the agent loads configuration from `/etc/sensu/agent.yml` by default.

To start the agent using a service manager:

Linux

```
sudo service sensu-agent start
```

Stopping the service

To stop the agent service using a service manager:

Linux

```
sudo service sensu-agent stop
```

Restarting 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
```

Enabling 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-server on` to enable the agent and `sudo chkconfig sensu-server off` to disable.

Getting service status

To see the status of the agent service using a service manager:

Linux

```
service sensu-agent status
```

Getting service version

To get the current agent version using the `sensu-agent` tool:

```
sensu-agent version
```

Getting help

The `sensu-agent` tool provides general and command-specific help flags:

```
# Show sensu-agent commands
sensu-agent help

# Show options for the sensu-agent start subcommand
sensu-agent start --help
```

Clustering

Agents can connect to a Sensu cluster by specifying any Sensu backend URL in the cluster in the `backend-url` configuration flag. For more information about clustering, see Sensu backend datastore configuration flags and the guide to running a Sensu cluster.

Time synchronization

System clocks between agents and the backend should be synchronized to a central NTP server. Out of sync system time may cause issues with keepalive, metric, and check alerts.

Registration

In practice, agent registration happens when a Sensu backend processes an agent keepalive event for an agent that is not already registered in the Sensu agent registry (based on the configured agent `name`).This agent registry is stored in the Sensu backend, and 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 process 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.

Registration events are useful for executing one-time handlers for new Sensu agents.For example, registration event handlers can be used to update external configuration management databases (CMDBs) such as ServiceNow.

To configure a registration event handler, please refer to the Sensu event handler documentation for instructions on creating a handler named `registration`.

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

Similarly to 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.

Configuration

You can specify the agent configuration using a `/etc/sensu/agent.yml` file or using `sensu-agent start` command-line flags.See the example config file provided with Sensu at `/usr/share/doc/sensu-go-agent-5.0.0/agent.yml.example`.Configuration provided via command-line flags overrides attributes specified in a configuration file.The agent loads configuration upon startup, so you must restart the agent for any configuration updates to take effect.

Configuration summary

```
$ sensu-agent start --help
start the sensu agent

Usage:
  sensu-agent start [flags]

Flags:
      --api-host string                address to bind the Sensu client HTTP
API to (default "127.0.0.1")
      --api-port int                   port the Sensu client HTTP API listens
on (default 3031)
      --backend-url strings            ws/wss URL of Sensu backend server (to
specify multiple backends use this flag multiple times) (default
[ws://127.0.0.1:8081])
      --cache-dir string               path to store cached data (default
"/var/cache/sensu/sensu-agent")
  -c, --config-file string             path to sensu-agent config file
      --deregister                     ephemeral agent
      --deregistration-handler string  deregistration handler that should
process the entity deregistration event.
      --disable-api                    disable the Agent HTTP API
      --disable-sockets                disable the Agent TCP and UDP event
sockets
  -h, --help                           help for start
      --keepalive-interval int         number of seconds to send between
keepalive events (default 20)
      --keepalive-timeout uint32       number of seconds until agent is
considered dead by backend (default 120)
      --labels stringToString          entity labels map (default [])
      --log-level string               logging level [panic, fatal, error,
warn, info, debug] (default "warn")
      --name string                    agent name (defaults to hostname)
(default "sensu-go-sandbox")
      --namespace string               agent namespace (default "default")
      --password string                agent password (default "P@ssw0rd!")
      --redact string                  comma-delimited customized list of fields
to redact
      --socket-host string             address to bind the Sensu client socket
to (default "127.0.0.1")
      --socket-port int                port the Sensu client socket listens on
```

```
(default 3030)
      --statsd-disable                    disables the statsd listener and metrics
server
      --statsd-event-handlers strings     comma-delimited list of event handlers
for statsd metrics
      --statsd-flush-interval int         number of seconds between statsd flush
(default 10)
      --statsd-metrics-host string        address used for the statsd metrics
server (default "127.0.0.1")
      --statsd-metrics-port int           port used for the statsd metrics server
(default 8125)
      --subscriptions string              comma-delimited list of agent
subscriptions
      --user string                       agent user (default "agent")
```

General configuration flags

backend-url

description	ws or wss URL of the Sensu backend server. To specify multiple backends using `sensu-agent start`, use this flag multiple times.
type	List
default	`ws://127.0.0.1:8081`
example	

```
# Command line examples
sensu-agent start --backend-url ws://0.0.0.0:8081
sensu-agent start --backend-url ws://0.0.0.0:8081 --
backend-url ws://0.0.0.0:8082

# /etc/sensu/agent.yml example
backend-url:
  - "ws://0.0.0.0:8081"
  - "ws://0.0.0.0:8082"
```

cache-dir

description	Path to store cached data
type	String
default	☐ Linux: `/var/cache/sensu/sensu-agent` ☐ Windows: `C:\\ProgramData\sensu\cache\sensu-agent`
example	

```
# Command line example
sensu-agent start --cache-dir /cache/sensu-agent

# /etc/sensu/agent.yml example
cache-dir: "/cache/sensu-agent"
```

config-file

description	Path to Sensu agent config file
type	String
default	☐ Linux: `/etc/sensu/agent.yml` ☐ FreeBSD: `/usr/local/etc/sensu/agent.yml` ☐ Windows: `C:\\ProgramData\sensu\config\agent.yml`
example	

```
# Command line example
sensu-agent start --config-file /sensu/agent.yml
sensu-agent start -c /sensu/agent.yml

# /etc/sensu/agent.yml example
config-file: "/sensu/agent.yml"
```

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize entities into meaningful collections that can be selected using filters and tokens.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	

```
# Command line example
sensu-agent start --labels proxy_type=website

# /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`
example	

```
# 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	`warn`
example	

```
# Command line example
sensu-agent start --log-level debug

# /etc/sensu/agent.yml example
log-level: "debug"
```

subscriptions

description	An array of agent subscriptions which determine which monitoring checks are executed by the agent. The subscriptions array items must be strings.
type	List
example	

```
# Command line examples
sensu-agent start --subscriptions disk-checks,process-
checks
sensu-agent start --subscriptions disk-checks --
subscriptions process-checks

# /etc/sensu/agent.yml example
subscriptions:
  - disk-checks
  - process-checks
```

API configuration flags

api-host

description	Bind address for the Sensu agent HTTP API
type	String
default	`127.0.0.1`
example	

```
# Command line example
sensu-agent start --api-host 0.0.0.0

# /etc/sensu/agent.yml example
api-host: "0.0.0.0"
```

api-port

description	Listening port for the Sensu agent HTTP API
type	Integer
default	`3031`
example	

```
# Command line example
sensu-agent start --api-port 4041

# /etc/sensu/agent.yml example
api-port: 4041
```

disable-api

description	Disable the agent HTTP API
type	Boolean
default	`false`

example	`# Command line example`
	`sensu-agent start --disable-api`
	
	`# /etc/sensu/agent.yml example`
	`disable-api: true`

Ephemeral agent configuration flags

deregister	
description	Indicates whether a deregistration event should be created upon Sensu agent process stop
type	Boolean
default	`false`

example	`# Command line example`
	`sensu-agent start --deregister`
	
	`# /etc/sensu/agent.yml example`
	`deregister: true`

deregistration-handler

description	The name of a deregistration handler that processes agent deregistration events. This flag overrides any handlers applied by the `deregistration-handler` backend configuration flag.
type	String
example	``` # Command line example sensu-agent start --deregistration-handler deregister # /etc/sensu/agent.yml example deregistration-handler: "deregister" ```

Keepalive configuration flags

keepalive-interval	
description	Number of seconds between keepalive events
type	Integer
default	`20`
example	``` # Command line example sensu-agent start --keepalive-interval 30 # /etc/sensu/agent.yml example keepalive-interval: 30 ```

keepalive-timeout	
description	Number of seconds after a missing keepalive event until the agent is

	considered unresponsive by the Sensu backend
type	Integer
default	`120`

example	`# Command line example`
	`sensu-agent start --keepalive-timeout 300`
	
	`# /etc/sensu/agent.yml example`
	`keepalive-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`

example	`# 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`
example	

```
# Command line example
sensu-agent start --user agent-01

# /etc/sensu/agent.yml example
user: "agent-01"
```

password

description	Sensu RBAC password used by the agent
type	String
default	`P@ssw0rd!`
example	

```
# Command line example
sensu-agent start --password secure-password

# /etc/sensu/agent.yml example
password: "secure-password"
```

redact

description	List of fields to redact when logging and sending keepalives
type	List
default	By default, Sensu redacts the following fields: `password`, `passwd`, `pass` `api_key` `api_token` `access_key` `secret_key`

	`private_key`, `secret`

example	`# Command line example`
	`sensu-agent start --redact secret,ec2_access_key`
	
	`# /etc/sensu/agent.yml example`
	`redact:`
	`  - secret`
	`  - ec2_access_key`

Socket configuration flags

socket-host

description	Address to bind the Sensu agent socket to
type	String
default	`127.0.0.1`

example	`# Command line example`
	`sensu-agent start --socket-host 0.0.0.0`
	
	`# /etc/sensu/agent.yml example`
	`socket-host: "0.0.0.0"`

socket-port

description	Port the Sensu agent socket listens on
type	Integer

default	`3030`
example	

```
# Command line example
sensu-agent start --socket-port 4030

# /etc/sensu/agent.yml example
socket-port: 4030
```

disable-sockets

description	Disable the agent TCP and UDP event sockets
type	Boolean
default	`false`
example	

```
# Command line example
sensu-agent start --disable-sockets

# /etc/sensu/agent.yml example
disable-sockets: true
```

StatsD configuration flags

statsd-disable

description	Disables the StatsD listener and metrics server
type	Boolean
default	`false`

example

```
# Command line example
sensu-agent start --statsd-disable

# /etc/sensu/agent.yml example
statsd-disable: true
```

statsd-event-handlers

description	List of event handlers for StatsD metrics
type	List

example

```
# Command line examples
sensu-agent start --statsd-event-handlers
influxdb,opentsdb
sensu-agent start --statsd-event-handlers influxdb --
statsd-event-handlers opentsdb

# /etc/sensu/agent.yml example
statsd-event-handlers:
  - influxdb
  - opentsdb
```

statsd-flush-interval

description	Number of seconds between StatsD flush
type	Integer
default	`10`

example

```
# Command line example
sensu-agent start --statsd-flush-interval 30

# /etc/sensu/agent.yml example
statsd-flush-interval: 30
```

statsd-metrics-host

description	Address used for the StatsD metrics server
type	String
default	`127.0.0.1`

example

```
# Command line example
sensu-agent start --statsd-metrics-host 0.0.0.0

# /etc/sensu/agent.yml example
statsd-metrics-host: "0.0.0.0"
```

statsd-metrics-port

description	Port used for the StatsD metrics server
type	Integer
default	`8125`

example

```
# Command line example
sensu-agent start --statsd-metrics-port 6125

# /etc/sensu/agent.yml example
```

```
statsd-metrics-port: 6125
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Assets

You can discover, download, and share assets using Bonsai, the Sensu asset index.Read the guide to using assets to get started.

What is an asset?

Assets are shareable, reusable packages that make it easy to deploy Sensu plugins.You can use assets to provide the plugins, libraries, and runtimes you need to automate your monitoring workflows.Sensu supports runtime assets for checks, filters, mutators, and handlers.

How do assets work?

Assets can be executed by the backend (for handler, filter, and mutator assets), orby the agent (for check assets). At runtime, the entity sequentially fetchesassets and stores them in its local cache. Asset dependencies are theninjected into the `PATH` so they are available when the command is executed.Subsequent check, handler, filter, or mutator executions look for the asset in the localcache and ensure the contents match the checksum. The backend or agent's local cache canbe set using the `--cache-dir` flag.

Asset format specification

Sensu expects an asset to be a tar archive (optionally gzipped) containing one or more executables within a bin folder.Any scripts or executables should be within a `bin/` folder within in the archive.See the Sensu Go Plugin template for an example asset and Bonsai configuration.

The following are injected into the execution context:

- `{PATH_TO_ASSET}/bin` is injected into the `PATH` environment variable.
- `{PATH_TO_ASSET}/lib` is injected into the `LD_LIBRARY_PATH` environmentvariable.
- `{PATH_TO_ASSET}/include` is injected into the `CPATH` environment variable.

Default cache directory

system	sensu-backend	sensu-agent
default	`/var/cache/sensu/sensu-backend`	`/var/cache/sensu/sensu-agent`
Windows	`C:\\ProgramData\sensu\cache\sensu-backend`	`C:\\ProgramData\sensu\cache\sensu-agent`

If the requested asset is not in the local cache, it is downloaded from the assetURL. The Sensu backend does not currently provide any storage for assets; theyare expected to be retrieved over HTTP or HTTPS.

Example structure

```
sensu-example-handler_1.0.0_linux_amd64
├── CHANGELOG.md
├── LICENSE
├── README.md
└── bin
  └── my-check.sh
└── lib
└── include
```

Asset specification

Top-level attributes

type

description	Top-level attribute specifying the `sensuctl create` resource type. Assets should always be of type `Asset`.
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "Asset"`

api_version

description	Top-level attribute specifying the Sensu API group and version. For assets in Sensu backend version 5.3, this attribute should always be `core/v2`.
required	Required for asset definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

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

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

spec

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

```
"spec": {
  "url": "http://example.com/asset.tar.gz",
  "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d1548
  "filters": [
```

```
    "entity.system.os == 'linux'",
    "entity.system.arch == 'amd64'"
  ]
}
```

Spec attributes

url

description	The URL location of the asset.
required	true
type	String
example	`"url": "http://example.com/asset.tar.gz"`

sha512

description	The checksum of the asset.
required	true
type	String
example	`"sha512": "4f926bf4328..."`

filters

description	A set of Sensu query expressions used by the agent to determine if the asset should be installed. If multiple expressions are included, each expression must return true in order for the agent to install the asset.
required	false
type	Array
example	`"filters": ["entity.system.os=='linux'", "entity.system.arch=='amd64'"]`

Metadata attributes

name	
description	The unique name of the asset, validated with Go regex `\A[\w\.\-]+\z`.
required	true
type	String
example	`"name": "check_script"`

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

example	`"namespace": "production"`

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize assets into meaningful collections that can be selected using filters and tokens.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	`"labels": { "environment": "development", "region": "us-west-2" }`

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are *not* used internally by Sensu and cannot be used to identify assets. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example

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

Examples

Minimum required asset attributes

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check_script",
    "namespace": "default"
  },
  "spec": {
    "url": "http://example.com/asset.tar.gz",
    "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4
  }
}
```

Asset definition

```
{
  "type": "Asset",
  "api_version": "core/v2",
  "metadata": {
    "name": "check script",
```

```
    "namespace": "default",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "slack-channel" : "#monitoring"
    }
  },
  "spec": {
    "url": "http://example.com/asset.tar.gz",
    "sha512":
"4f926bf4328fbad2b9cac873d117f771914f4b837c9c85584c38ccf55a3ef3c2e8d154812246e5dda4
    "filters": [
      "entity.system.os == 'linux'",
      "entity.system.arch == 'amd64'"
    ]
  }
}
```

Sharing an asset on Bonsai

Share your open-source assets on Bonsai and connect with the Sensu Community.Bonsai supports assets hosted on GitHub and released using GitHub releases.For more information about creating Sensu Plugins, see the Sensu Plugin specification.

Bonsai requires a `bonsai.yml` configuration file in the root directory of your repository that includes the project description, platforms, asset filenames, and SHA-512 checksums.For a Bonsai-compatible asset template using Go and GoReleaser, see the Sensu Go plugin skeleton.

To share your asset on Bonsai, log in to Bonsai with your GitHub account and authorize Sensu.Once logged in, you can register your asset on Bonsai by adding the GitHub repository, description, and tags. Make sure to provide a helpful README for your asset with configuration examples.

`bonsai.yml` example

```
---
```

```
description: "#{repo}"
builds:
- platform: "linux"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'linux'"
  -  "entity.system.arch == 'amd64'"

- platform: "Windows"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_windows_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'windows'"
  -  "entity.system.arch == 'amd64'"
```

`bonsai.yml` specification

description	
description	The project description
required	true
type	String
example	`description: "#{repo}"`

builds	
description	An array of asset details per platform
required	true

type	Array
example	

```
builds:
- platform: "linux"
  arch: "amd64"
  asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"
  sha_filename: "#{repo}_#{version}_sha512-checksums.txt"
  filter:
  -  "entity.system.os == 'linux'"
  -  "entity.system.arch == 'amd64'"
```

Builds specification

platform

description	The platform supported by the asset
required	true
type	String
example	

```
- platform: "linux"
```

arch

description	The architecture supported by the asset
required	true
type	String

example	`arch: "amd64"`

asset_filename

description	The filename of the archive containing the asset
required	true
type	String
example	`asset_filename: "#{repo}_#{version}_linux_amd64.tar.gz"`

sha_filename

description	The SHA-512 checksum for the asset archive
required	true
type	String
example	`sha_filename: "#{repo}_#{version}_sha512-checksums.txt"`

filter

description	Entity filters specifying the operating system and architecture supported by the asset
required	false
type	Array

example

```
filter:
- "entity.system.os == 'linux'"
- "entity.system.arch == 'amd64'"
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Sensu backend

The Sensu backend is a service that manages check requests and event data.Every Sensu backend includes an integrated transport for scheduling checks using subscriptions, an event processing pipeline that applies filters, mutators, and handlers, an embedded etcd datastore for storing configuration and state, a Sensu API, Sensu dashboard, and `sensu-backend` command-line tool.The Sensu backend is available for Ubuntu/Debian and RHEL/CentOS distributions of Linux.See the installation guide to install the backend.

Event pipeline

The backend processes event data and executes filters, mutators, and handlers.These pipelines are powerful tools to automate your monitoring workflows.To learn more about filters, mutators, and

handlers, see:

- Guide to sending Slack alerts with handlers
- Guide to reducing alerting fatigue with filters
- Filters reference documentation
- Mutators reference documentation
- Handlers reference documentation

Check scheduling

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:

- Guide to monitoring server resources with checks
- Guide to collecting metrics with checks
- Checks reference documentation

Operation

NOTE: Commands in this section may require administrative privileges.

Starting the service

Use the `sensu-backend` tool to start the backend and apply configuration flags.

To start the backend with configuration flags:

```
sensu-backend start --state-dir /var/lib/sensu/sensu-backend --log-level debug
```

To see available configuration flags and defaults:

```
sensu-backend start --help
```

If no configuration flags are provided, the backend loads configuration from `/etc/sensu/backend.yml` by default.

To start the backend using a service manager:

```
service sensu-backend start
```

Stopping the service

To stop the backend service using a service manager:

```
service sensu-backend stop
```

Restarting the service

You must restart the backend to implement any configuration updates.

To restart the backend using a service manager:

```
service sensu-backend restart
```

Enabling 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.

Getting service status

To see the status of the backend service using a service manager:

```
service sensu-backend status
```

Getting service version

To get the current backend version using the `sensu-backend` tool:

```
sensu-backend version
```

Getting help

The `sensu-backend` tool provides general and command-specific help flags:

```
# Show sensu-backend commands
sensu-backend help

# Show options for the sensu-backend start subcommand
sensu-backend start --help
```

Clustering

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 clustering system used by etcd.Clustering lets you synchronize data between backends and get the benefits of a highly available configuration.To configure a cluster, see:

- Datastore configuration flags
- Guide to running a Sensu cluster

Time synchronization

System clocks between agents and the backend should be synchronized to a central NTP server. Out of sync system time may cause issues with keepalive, metric, and check alerts.

Configuration

You can specify the backend configuration using a `/etc/sensu/backend.yml` file or using `sensu-backend start` configuration flags.The backend requires that the `state-dir` flag be set before starting; all other required flags have default values.See the example config file provided with Sensu at `/usr/share/doc/sensu-go-backend-5.0.0/backend.yml.example`.The backend loads configuration upon startup, so you must restart the backend for any configuration updates to take effect.

Configuration summary

```
$ sensu-backend start --help
start the sensu backend

Usage:
  sensu-backend start [flags]

General Flags:
      --agent-host string               agent listener host (default "[::]")
      --agent-port int                  agent listener port (default 8081)
      --api-listen-address string       address to listen on for api traffic
(default "[::]:8080")
```

```
      --api-url string                  url of the api to connect to (default
"http://localhost:8080")
      --cache-dir string                path to store cached data (default
"/var/cache/sensu/sensu-backend")
      --cert-file string                tls certificate
  -c, --config-file string              path to sensu-backend config file
      --dashboard-host string           dashboard listener host (default "[::]")
      --dashboard-port int              dashboard listener port (default 3000)
      --debug                           enable debugging and profiling features
      --deregistration-handler string   default deregistration handler
  -h, --help                            help for start
      --insecure-skip-tls-verify        skip ssl verification
      --key-file string                 tls certificate key
      --log-level string                logging level [panic, fatal, error,
warn, info, debug] (default "warn")
  -d, --state-dir string                path to sensu state storage (default
"/var/lib/sensu")
      --trusted-ca-file string          tls certificate authority

Store Flags:
      --etcd-advertise-client-urls strings         list of this member's client
URLs to advertise to the rest of the cluster. (default [http://localhost:2379])
      --etcd-cert-file string                      path to the client server TLS
cert file
      --etcd-client-cert-auth                      enable client cert
authentication
      --etcd-initial-advertise-peer-urls strings   list of this member's peer
URLs to advertise to the rest of the cluster (default [http://127.0.0.1:2380])
      --etcd-initial-cluster string                initial cluster configuration
for bootstrapping (default "default=http://127.0.0.1:2380")
      --etcd-initial-cluster-state string          initial cluster state ("new"
or "existing") (default "new")
      --etcd-initial-cluster-token string          initial cluster token for the
etcd cluster during bootstrap
      --etcd-key-file string                       path to the client server TLS
key file
      --etcd-listen-client-urls strings            list of URLs to listen on for
client traffic (default [http://127.0.0.1:2379])
      --etcd-listen-peer-urls strings              list of URLs to listen on for
peer traffic (default [http://127.0.0.1:2380])
```

```
      --etcd-name string                          human-readable name for this
member (default "default")
      --etcd-peer-cert-file string                path to the peer server TLS
cert file
      --etcd-peer-client-cert-auth                enable peer client cert
authentication
      --etcd-peer-key-file string                 path to the peer server TLS
key file
      --etcd-peer-trusted-ca-file string          path to the peer server TLS
trusted CA file
      --etcd-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

cache-dir

description	Path to store cached data
type	String
default	Linux: `/var/cache/sensu/sensu-backend` Windows: `C:\\ProgramData\sensu\cache\sensu-backend`

example

```
# Command line example
sensu-backend start --cache-dir /cache/sensu-backend

# /etc/sensu/backend.yml example
cache-dir: "/cache/sensu-backend"
```

config-file

description	Path to Sensu backend config file
type	String
default	Linux: `/etc/sensu/backend.yml` FreeBSD: `/usr/local/etc/sensu/backend.yml` Windows: `C:\\ProgramData\sensu\config\backend.yml`
example	

```
# Command line example
sensu-backend start --config-file /etc/sensu/backend.yml
sensu-backend start -c /etc/sensu/backend.yml

# /etc/sensu/backend.yml example
config-file: "/etc/sensu/backend.yml"
```

debug

description	Enable debugging and profiling features
type	Boolean
default	`false`
example	

```
# Command line example
sensu-backend start --debug

# /etc/sensu/backend.yml example
debug: true
```

deregistration-handler

description	Default event handler to use when processing agent deregistration

	events.
type	String
default	`""`

example	`# Command line example`
	`sensu-backend start --deregistration-handler`
	`/path/to/handler.sh`
	
	`# /etc/sensu/backend.yml example`
	`deregistration-handler: "/path/to/handler.sh"`

log-level

description	Logging level: `panic`, `fatal`, `error`, `warn`, `info`, or `debug`
type	String
default	`warn`

example	`# Command line example`
	`sensu-backend start --log-level debug`
	
	`# /etc/sensu/backend.yml example`
	`log-level: "debug"`

state-dir

description	Path to Sensu state storage: `/var/lib/sensu/sensu-backend` for Linux and `C:\\ProgramData\sensu\data` for Windows.
type	String

required	true
example	

```
# Command line example
sensu-backend start --state-dir /var/lib/sensu/sensu-backend
sensu-backend start -d /var/lib/sensu/sensu-backend

# /etc/sensu/backend.yml example
state-dir: "/var/lib/sensu/sensu-backend"
```

api-listen-address

description	Address the API daemon will listen for requests on
type	String
default	`[::]:8080`
example	

```
# Command line example
sensu-backend start --api-listen-address [::]:8080

# /etc/sensu/backend.yml example
api-listen-address: "[::]:8080"
```

api-url

description	URL used to connect to the API
type	String
default	`http://localhost:8080`
example	

```
# Command line example
sensu-backend start --api-url http://localhost:8080

# /etc/sensu/backend.yml example
api-url: "http://localhost:8080"
```

Agent communication configuration flags

agent-host	
description	agent listener host, listens on all IPv4 and IPv6 addresses by default
type	String
default	`[::]`
example	

```
# Command line example
sensu-backend start --agent-host 127.0.0.1

# /etc/sensu/backend.yml example
agent-host: "127.0.0.1"
```

agent-port	
description	agent listener port
type	Integer
default	`8081`
example	

```
# Command line example
sensu-backend start --agent-port 8081
```

```
# /etc/sensu/backend.yml example
agent-port: 8081
```

Security configuration flags

cert-file

description	Path to the primary backend certificate file. This certificate secures communications between Sensu Dashboard and end user web browsers, as well as communication between sensuctl and the Sensu API.
type	String
default	`""`

example	`# Command line example`
	`sensu-backend start --cert-file /path/to/ssl/cert.pem`
	
	`# /etc/sensu/backend.yml example`
	`cert-file: "/path/to/ssl/cert.pem"`

key-file

description	SSL/TLS certificate key. This key secures communication with the Sensu Dashboard and API.
type	String
default	`""`
example	`# Command line example`

```
sensu-backend start --key-file /path/to/ssl/key.pem

# /etc/sensu/backend.yml example
key-file: "/path/to/ssl/key.pem"
```

trusted-ca-file

description	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.
type	String
default	`""`
example	

```
# Command line example
sensu-backend start --trusted-ca-file /path/to/trusted-certificate-authorities.pem

# /etc/sensu/backend.yml example
trusted-ca-file: "/path/to/trusted-certificate-authorities.pem"
```

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`
example	

```
# Command line example
sensu-backend start --insecure-skip-tls-verify

# /etc/sensu/backend.yml example
insecure-skip-tls-verify: true
```

Dashboard configuration flags

dashboard-host

description	Dashboard listener host
type	String
default	`[::]`
example	

```
# Command line example
sensu-backend start --dashboard-host 127.0.0.1

# /etc/sensu/backend.yml example
dashboard-host: "127.0.0.1"
```

dashboard-port

description	Dashboard listener port
type	Integer
default	`3000`
example	

```
# Command line example
sensu-backend start --dashboard-port 4000
```

```
# /etc/sensu/backend.yml example
dashboard-port: 4000
```

Datastore and cluster configuration flags

etcd-advertise-client-urls

description	List of this member's client URLs to advertise to the rest of the cluster.
type	List
default	`http://localhost:2379`
example	

```
# Command line examples
sensu-backend start --etcd-advertise-client-urls
http://localhost:2378,http://localhost:2379
sensu-backend start --etcd-advertise-client-urls
http://localhost:2378 --etcd-advertise-client-urls
http://localhost:2379

# /etc/sensu/backend.yml example
etcd-advertise-client-urls:
  - http://localhost:2378
  - http://localhost:2379
```

etcd-cert-file

description	Path to the etcd client API TLS cert file. Secures communication between the embedded etcd client API and any etcd clients.
type	String

default	`""`

example	`# Command line example`
	`sensu-backend start --etcd-cert-file ./client.pem`
	
	`# /etc/sensu/backend.yml example`
	`etcd-cert-file: "./client.pem"`

etcd-client-cert-auth

description	Enable client cert authentication
type	Boolean
default	`false`

example	`# Command line example`
	`sensu-backend start --etcd-client-cert-auth`
	
	`# /etc/sensu/backend.yml example`
	`etcd-client-cert-auth: true`

etcd-initial-advertise-peer-urls

description	List of this member's peer URLs to advertise to the rest of the cluster
type	List
default	`http://127.0.0.1:2380`
example	

```
# Command line examples
sensu-backend start --etcd-listen-peer-urls
https://10.0.0.1:2380,https://10.1.0.1:2380
sensu-backend start --etcd-listen-peer-urls
https://10.0.0.1:2380 --etcd-listen-peer-urls
https://10.1.0.1:2380

# /etc/sensu/backend.yml example
etcd-listen-peer-urls:
  - https://10.0.0.1:2380
  - https://10.1.0.1:2380
```

etcd-initial-cluster

description	Initial cluster configuration for bootstrapping
type	String
default	`http://127.0.0.1:2380`

example

```
# Command line example
sensu-backend start --etcd-initial-cluster backend-
0=https://10.0.0.1:2380,backend-
1=https://10.1.0.1:2380,backend-2=https://10.2.0.1:2380

# /etc/sensu/backend.yml example
etcd-initial-cluster: "backend-
0=https://10.0.0.1:2380,backend-
1=https://10.1.0.1:2380,backend-2=https://10.2.0.1:2380"
```

etcd-initial-cluster-state

description Initial cluster state (`new` or `existing`)

type	String
default	`new`
example	

```
# Command line example
sensu-backend start --etcd-initial-cluster-state
existing

# /etc/sensu/backend.yml example
etcd-initial-cluster-state: "existing"
```

etcd-initial-cluster-token

description	Initial cluster token for the etcd cluster during bootstrap
type	String
default	`""`
example	

```
# Command line example
sensu-backend start --etcd-initial-cluster-token sensu

# /etc/sensu/backend.yml example
etcd-initial-cluster-token: "sensu"
```

etcd-key-file

description	Path to the etcd client API TLS key file. Secures communication between the embedded etcd client API and any etcd clients.
type	String

example	``` # Command line example sensu-backend start --etcd-key-file ./client-key.pem # /etc/sensu/backend.yml example etcd-key-file: "./client-key.pem" ```

etcd-listen-client-urls

description	List of URLs to listen on for client traffic
type	List
default	`http://127.0.0.1:2379`

example	```
	# Command line examples
	sensu-backend start --etcd-listen-client-urls
	https://10.0.0.1:2379,https://10.1.0.1:2379
	sensu-backend start --etcd-listen-client-urls
	https://10.0.0.1:2379 --etcd-listen-client-urls
	https://10.1.0.1:2379
	
	# /etc/sensu/backend.yml example
	etcd-listen-client-urls:
	  - https://10.0.0.1:2379
	  - https://10.1.0.1:2379
	``

etcd-listen-peer-urls

description	List of URLs to listen on for peer traffic
type	List

default	`http://127.0.0.1:2380`

example

```
# Command line examples
sensu-backend start --etcd-listen-peer-urls
https://10.0.0.1:2380,https://10.1.0.1:2380
sensu-backend start --etcd-listen-peer-urls
https://10.0.0.1:2380 --etcd-listen-peer-urls
https://10.1.0.1:2380

# /etc/sensu/backend.yml example
etcd-listen-peer-urls:
  - https://10.0.0.1:2380
  - https://10.1.0.1:2380
```

etcd-name

description	Human-readable name for this member
type	String
default	`default`

example

```
# Command line example
sensu-backend start --etcd-name backend-0

# /etc/sensu/backend.yml example
etcd-name: "backend-0"
```

etcd-peer-cert-file

description	Path to the peer server TLS certificate file. This certificate secures communication between etcd cluster members.

type	String
example	``` # Command line example sensu-backend start --etcd-peer-cert-file ./backend-0.pem # /etc/sensu/backend.yml example etcd-peer-cert-file: "./backend-0.pem" ```

etcd-peer-client-cert-auth

description	Enable peer client cert authentication
type	Boolean
default	`false`
example	``` # Command line example sensu-backend start --etcd-peer-client-cert-auth # /etc/sensu/backend.yml example etcd-peer-client-cert-auth: true ```

etcd-peer-key-file

description	Path to the etcd peer API TLS key file. Secures communication between etcd cluster members.
type	String
example	``` # Command line example sensu-backend start --etcd-peer-key-file ./backend-0- ```

```
key.pem

# /etc/sensu/backend.yml example
etcd-peer-key-file: "./backend-0-key.pem"
```

etcd-peer-trusted-ca-file

description	Path to the etcd peer API server TLS trusted CA file. This certificate secures communication between etcd cluster members.
type	String
example	

```
# Command line example
sensu-backend start --etcd-peer-trusted-ca-file ./ca.pem

# /etc/sensu/backend.yml example
etcd-peer-trusted-ca-file: "./ca.pem"
```

etcd-trusted-ca-file

description	Path to the client server TLS trusted CA cert file. Secures communication with the etcd client server.
type	String
default	`""`
example	

```
# Command line example
sensu-backend start --etcd-trusted-ca-file ./ca.pem

# /etc/sensu/backend.yml example
etcd-trusted-ca-file: "./ca.pem"
```

no-embed-etcd	
description	Don't embed etcd, use external etcd instead
type	Boolean
default	`false`
example	

```
# Command line example
sensu-backend start --no-embed-etcd

# /etc/sensu/backend.yml example
no-embed-etcd: true
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Checks

How do checks work?

Check commands

Each Sensu check definition defines a **command** and the **interval** at whichit should be executed. Check commands are executable commands whichwill be executed by the Sensu agent.

A command may include command line arguments for controlling the behavior of thecommand executable. Most Sensu check plugins provide support for command linearguments for reusability.

Sensu advises against requiring root privileges to execute checkcommands or scripts. The Sensu user is not permitted to kill timed out processesinvoked by the root user, which could result in zombie processes.

How and where are check commands executed?

All check commands are executed by Sensu agents as the `sensu` user. Commandsmust be executable files that are discoverable on the Sensu agent system (ex:installed in a system `$PATH` directory).

Check scheduling

Checks are exclusively scheduled by the Sensu backend, which schedules andpublishes check execution requests to entities via a Publish/Subscribemodel .

Checks have a defined set of subscribers, a list of transporttopics that check requests will be published to. Sensu entities becomesubscribers to these topics (called subscriptions) via their individualsubscriptions attribute. In practice, subscriptions will typically correspond toa specific role and/or responsibility (ex: a webserver or database).

Subscriptions are a powerful primitives in the monitoring context because theyallow you to effectively monitor for specific behaviors or characteristicscorresponding to the function being provided by a particular system. Forexample, disk capacity thresholds might be more important (or at leastdifferent) on a database server as opposed to a webserver; conversely, CPUand/or memory usage thresholds might be more important on a caching system thanon a file server. Subscriptions also allow you to configure check requests foran entire group or subgroup of systems rather than require a traditional 1:1 mapping.

Checks can be scheduled in an interval or cron fashion. It's important to notethat for interval checks, an initial offset is calculated to splay the check's *first* scheduled request. This helps to balance the load of both the backendand the agent, and may result in a delay before initial check execution.

Check result specification

Although the Sensu agent will attempt to execute anycommand defined for a check, successful processing of check results requiresadherence to a simple specification.

- Result data is output to STDOUT or STDERR
  - For standard checks this output is typically a human-readable message
  - For metrics checks this output contains the measurements gathered by thecheck
- Exit status code indicates state
  - `0` indicates "OK"
  - `1` indicates "WARNING"

- `2` indicates “CRITICAL”
- exit status codes other than `0`, `1`, or `2` indicate an “UNKNOWN” orcustom status

PRO TIP: Those familiar with the **Nagios** *monitoringsystem may recognize this specification, as it is the same one used by Nagiosplugins. As a result, Nagios plugins can be used with Sensu without anymodification.*

At every execution of a check command – regardless of success or failure – theSensu agent publishes the check’s result for eventual handling by the **eventprocessor** (the Sensu backend).

Check token substitution

Sensu check definitions may include attributes that you may wish to override onan entity-by-entity basis. For example, check commands – which may includecommand line arguments for controlling the behavior of the check command – maybenefit from entity-specific thresholds, etc. Sensu check tokens are checkdefinition placeholders that will be replaced by the Sensu agent with the corresponding entity definition attributes values (including custom attributes).

Learn how to use check tokens with the Sensu tokens referencedocumentation.

NOTE: Check tokens are processed before check execution, therefore 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 ofcheck command execution. The Sensu agent will execute the appropriate configuredhook command, depending on the check execution status (ex: 0, 1, 2).

Learn how to use check hooks with the Sensu hooks referencedocumentation.

Proxy requests

Sensu supports running checks where the results are considered to be for anentity that isn’t actually the one executing the check, regardless of whetherthat entity is a Sensu agent entity or a **proxy entity**.Proxy entities allow Sensu to monitor external resourceson systems or devices where a Sensu agent cannot be installed, like anetwork switch or a website.

By specifying the proxy_requests attributes in a check, Sensu runs the checkfor each entity that matches certain definitions specified in the `entity_attributes`.The attributes supplied must match exactly as stated; no variables or directives haveany special meaning, but you can still use Sensu query expressions toperform more complicated filtering on the available value, such as findingentities with particular subscriptions.

Check specification

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Checks should always be of type `CheckConfig`.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "CheckConfig"`

api_version	
description	Top-level attribute specifying the Sensu API group and version. For checks in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the check, including the `name` and `namespace` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the check definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. See the metadata attributes reference for details.
required	Required for check definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	`"metadata": { "name": "collect-metrics", "namespace": "default", "labels": { "region": "us-west-1" }, "annotations": { "slack-channel" : "#monitoring" } }`

spec

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

example	<pre>"spec": {
	  "command": "/etc/sensu/plugins/check-chef-client.go",
	  "interval": 10,
	  "publish": true,
	  "subscriptions": [
	  "production"
	  ]
	}</pre>

Spec attributes

command

description	The check command to be executed.
required	true
type	String
example	<pre>"command": "/etc/sensu/plugins/check-chef-client.go"</pre>

subscriptions

description	An array of Sensu entity subscriptions that check requests will be sent to. The array cannot be empty and its items must each be a string.
required	true
type	Array
example	<pre>"subscriptions": ["production"]</pre>

handlers

description	An array of Sensu event handlers (names) to use for events created by the check. Each array item must be a string.
required	false
type	Array
example	`"handlers": ["pagerduty", "email"]`

interval

description	How often the check is executed, in seconds
required	true (unless `publish` is `false` or `cron` is configured)
type	Integer
example	`"interval": 60`

cron

description	When the check should be executed, using cron syntax or these predefined schedules.
required	true (unless `publish` is `false` or `interval` is configured)
type	String
example	

```
"cron": "0 0 * * *"
```

publish

description	If check requests are published for the check.
required	false
type	Boolean
example	`"publish": false`

timeout

description	The check execution duration timeout in seconds (hard stop).
required	false
type	Integer
example	`"timeout": 30`

ttl

description	The time to live (TTL) in seconds until check results are considered stale. 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 accommodate time for the check execution and result processing to complete. For example, if a check has an `interval` of `60` (seconds) and a `timeout` of `30` (seconds), an appropriate `ttl` would be a minimum

	of `90` (seconds).
required	false
type	Integer
example	`"ttl": 100`

stdin

description	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. This attribute cannot be used with existing Sensu check plugins, nor Nagios plugins etc, as Sensu agent will wait indefinitely for the check process to read and close STDIN.
required	false
type	Boolean
default	false
example	`"stdin": true`

low_flap_threshold

description	The flap detection low threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios.
required	false
type	Integer

example	`"low_flap_threshold": 20`

high_flap_threshold

description	The flap detection high threshold (% state change) for the check. Sensu uses the same flap detection algorithm as Nagios.
required	true (if `low_flap_threshold` is configured)
type	Integer
example	`"high_flap_threshold": 60`

runtime_assets

description	An array of Sensu assets (names), required at runtime for the execution of the `command`
required	false
type	Array
example	`"runtime_assets": ["ruby-2.5.0"]`

check_hooks

description	An array of check response types with respective arrays of Sensu hook names. Sensu hooks are commands run by the Sensu agent in response to the result of the check command execution. Hooks are executed, in order of precedence, based on their severity type: `1` to

	`255`, `ok`, `warning`, `critical`, `unknown`, and finally `non-zero`.
required	false
type	Array
example
```
"check_hooks": [
  {
    "0": [
      "passing-hook","always-run-this-hook"
    ]
  },
  {
    "critical": [
      "failing-hook","collect-diagnostics","always-run-this-hook"
    ]
  }
]
```
 |

proxy_entity_name

description	The entity name, used to create a proxy entity for an external resource (i.e., a network switch).
required	false
type	String
validated	`\A[\w\.\-]+\z`
example
```
"proxy_entity_name": "switch-dc-01"
```
 |

proxy_requests

description	Sensu proxy request attributes allow you to assign the check to run for multiple entities according to their `entity_attributes`. In the example below, the check executes for all entities with entity class `proxy` and the custom proxy type label `website`. Proxy requests are a great way to reuse check definitions for a group of entities. For more information, see the proxy requests specification and the guide to monitoring external resources.
required	false
type	Hash
example	

```
"proxy_requests": {
  "entity_attributes": [
    "entity.entity_class == 'proxy'",
    "entity.labels.proxy_type == 'website'"
  ],
  "splay": true,
  "splay_coverage": 90
}
```

silenced

description	If the event is to be silenced.
type	boolean
example	

```
"silenced": false
```

env_vars

description	An array of environment variables to use with command execution.

NOTE: To add `env_vars` to a check, use `sensuctl create`.

required	false
type	Array
example	`"env_vars": ["RUBY_VERSION=2.5.0", "CHECK_HOST=my.host.internal"]`

output_metric_format

description	The metric format generated by the check command. Sensu supports the following metric formats: `nagios_perfdata` (Nagios Performance Data) `graphite_plaintext` (Graphite Plaintext Protocol) `influxdb_line` (InfluxDB Line Protocol) `opentsdb_line` (OpenTSDB Data Specification) When a check includes an `output_metric_format`, Sensu will extract the metrics from the check output and add them to the event data in Sensu metric format. For more information about extracting metrics using Sensu, see the guide.
required	false
type	String
example	`"output_metric_format": "graphite_plaintext"`

output_metric_handlers

description	An array of Sensu handlers to use for events created by the check. Each array item must be a string. `output_metric_handlers` should be used in place of the `handlers` attribute if `output_metric_format` is configured.

Metric handlers must be able to process Sensu metric format. For an example, see the Sensu InfluxDB handler.

required	false
type	Array
example	`"output_metric_handlers": ["influx-db"]`

round_robin

description	Round-robin check subscriptions are not yet implemented in Sensu Go. Although the `round_robin` attribute appears in check definitions by default, it is a placeholder and should not be modified.
example	`"round_robin": false`

subdue

description	Check subdues are not yet implemented in Sensu Go. Although the `subdue` attribute appears in check definitions by default, it is a placeholder and should not be modified.
example	`"subdue": null`

Metadata attributes

name

description	A unique string used to identify the check. Check names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each check must have a unique name within its namespace.
required	true
type	String
example	`"name": "check-cpu"`

namespace

description	The Sensu RBAC namespace that this check belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize checks into meaningful collections that can be selected using filters and tokens.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.

default	`null`

example

```
"labels": {
  "environment": "development",
  "region": "us-west-2"
}
```

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are *not* used internally by Sensu and cannot be used to identify checks. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example

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

Proxy requests attributes

entity_attributes

description	Sensu entity attributes to match entities in the registry, using Sensu query expressions

required	false
type	Array

example	<pre>"entity_attributes": [
	  "entity.entity_class == 'proxy'",
	  "entity.labels.proxy_type == 'website'"
	]</pre>

splay

description	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. 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`s 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	<pre>"splay": true</pre>

splay_coverage

description	The **percentage** of the check interval over which Sensu can execute the check for all applicable entities, as defined in the entity attributes. Sensu uses the splay coverage attribute to determine the amount of time check requests can be published over (before the next check interval).

required	required if `splay` attribute is set to `true`
type	Integer
example	`"splay_coverage": 90`

Examples

Minimum recommended check attributes

NOTE: The attribute `interval` is not required if a valid `cron` schedule is defined.

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default",
    "name": "check_minimum"
  },
  "spec": {
    "command": "collect.sh",
    "subscriptions": [
      "system"
    ],
    "handlers": [
      "slack"
    ],
    "interval": 10,
    "publish": true
  }
}
```

Metric check

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "collect-metrics",
    "namespace": "default",
    "labels": {
      "region": "us-west-1"
    },
    "annotations": {
      "slack-channel" : "#monitoring"
    }
  },
  "spec": {
    "command": "collect.sh",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": null,
    "subscriptions": [
      "system"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "ttl": 0,
    "timeout": 0,
    "output_metric_format": "graphite_plaintext",
    "output_metric_handlers": [
      "influx-db"
    ],
    "env_vars": null
  }
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Entities

How do entities work?

Agent entities are monitoring agents, which are installed and run on every system that needs to be monitored. The entity is responsible for registering the system with the Sensu backend service, sending keepalive messages (the Sensu heartbeat mechanism), and executing monitoring checks. Each entity is a member of one or more `subscriptions` – a list of roles and/or responsibilities assigned to the agent entity (ex: a webserver or a database). Sensu entities will "subscribe" to (or watch for) check requests published by the Sensu server (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 server).

An entity represents anything (ex: server, container, network switch) that needs to be monitored, including the full range of infrastructure, runtime and application types that compose a complete monitoring environment (from server hardware to serverless functions).We call these monitored parts of an infrastructure “entities”.An entity not only provides context to event data (what/where the event is from) but an event’s uniqueness is determined by the check name and the name of the entity upon which the check ran.In addition, an entity can contain system information such as the hostname, OS, platform, and version.

Proxy entities

Proxy entities (formerly known as proxy clients, “Just-in-time” or “JIT” clients) are dynamically created entities, added to the entity store if an entity does not already exist for a check result. Proxy entity registration differs from keepalive-based registration because the registration event happens while processing a check result (not a keepalive message). Sensu proxy entities allow Sensu to monitor external resources on systems and/or devices where a sensu-agent cannot be installed (such a network switch) using the defined check ProxyEntityName to create a proxy entity for the external resource.

Managing entity labels

Custom labels let you organize entities into meaningful collections that can be selected using filters and tokens.

Proxy entities

For entities with class `proxy`, you can create and manage labels using sensuctl.For example, to create a proxy entity with a `url` label using sensuctl `create`, create a file called `example.json` with an entity definition that includes `labels`.

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-docs",
```

```
    "namespace": "default",
    "labels": {
      "url": "docs.sensu.io"
    }
  },
  "spec": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "proxy",
    "last_seen": 0,
    "subscriptions": [],
    "system": {
      "network": {
        "interfaces": null
      }
    }
  }
}
```

Then run `sensuctl create` to create the entity based on the definition.

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

To add a label to an existing entity, you can use sensuctl `edit`.For example, run `sensuctl edit` to add a `url` label to a `sensu-docs` entity.

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

And update the `metadata` scope to include `labels`.

```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "sensu-docs",
    "namespace": "default",
```

```
    "labels": {
      "url": "docs.sensu.io"
    }
  },
  "spec": {
    "...": "..."
  }
}
```

Agent entities

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

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

Or using `sensu-agent start` configuration flags.

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

Entities specification

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Entities should always be of type `Entity`.
required	Required for entity definitions in `wrapped-json` or `yaml` format for

	use with `sensuctl create`.
type	String
example	`"type": "Entity"`

api_version

description	Top-level attribute specifying the Sensu API group and version. For entities in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for entity definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

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

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

spec

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

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

```
            {
              "name": "enp0s3",
              "mac": "08:00:27:11:ad:d2",
              "addresses": [
                "10.0.2.15/24",
                "fe80::26a5:54ec:cf0d:9704/64"
              ]
            },
            {
              "name": "enp0s8",
              "mac": "08:00:27:bc:be:60",
              "addresses": [
                "172.28.128.3/24",
                "fe80::a00:27ff:febc:be60/64"
              ]
            }
          ]
        },
        "arch": "amd64"
      },
      "subscriptions": [
        "entity:webserver01"
      ],
      "last_seen": 1542667231,
      "deregister": false,
      "deregistration": {},
      "user": "agent",
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ]
    }
```

Spec attributes

entity_class

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

subscriptions

description	A list of subscription names for the entity. The entity by default has an entity-specific subscription, in the format of `entity:{name}` where `name` is the entity's hostname.
required	false
type	array
default	The entity-specific subscription.
example	`"subscriptions": ["web", "prod", "entity:example-entity"]`

system	
description	System information about the entity, such as operating system and platform. See the system attributes for more information.
required	false
type	map
example	


```
{
  "system": {
    "hostname": "example-hostname",
    "os": "linux",
    "platform": "ubuntu",
    "platform_family": "debian",
    "platform_version": "16.04",
    "network": {
      "interfaces": [
        {
          "name": "lo",
          "addresses": [
            "127.0.0.1/8",
            "::1/128"
          ]
        },
        {
          "name": "eth0",
          "mac": "52:54:00:20:1b:3c",
          "addresses": [
            "93.184.216.34/24",
            "2606:2800:220:1:248:1893:25c8:1946/10"
          ]
        }
      ]
    },
    "arch": "amd64"
  }
}
```

last_seen

description	Timestamp the entity was last seen, in seconds since the Unix epoch.
required	false
type	integer
example	`"last_seen": 1522798317`

deregister

description	If the entity should be removed when it stops sending keepalive messages.
required	false
type	boolean
default	false
example	`"deregister": false`

deregistration

description	A map containing a handler name, for use when an entity is deregistered. See the deregistration attributes for more information.
required	false
type	map

example

```
{
  "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”]

example

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

user

description	Sensu RBAC username used by the entity. Agent entities require get, list, create, update, and delete permissions for events across all namespaces.
type	String
default	`agent`

example	`"user": "agent"`

Metadata attributes

name	
description	The unique name of the entity, validated with Go regex `\A[\w\.\-]+\z`.
required	true
type	String
example	`"name": "example-hostname"`

namespace	
description	The Sensu RBAC namespace that this entity belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize entities into meaningful collections that can be selected using filters and tokens.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	`"labels": { "environment": "development", "region": "us-west-2" }`

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are *not* used internally by Sensu and cannot be used to identify entities. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	`"annotations": { "managed-by": "ops", "slack-channel": "#monitoring", "playbook": "www.example.url" }`

System attributes

hostname

description	The hostname of the entity.
required	false
type	string
example	`"hostname": "example-hostname"`

os

description	The entity's operating system.
required	false
type	string
example	`"os": "linux"`

platform

description	The entity's operating system distribution.
required	false
type	string
example	

```
"platform": "ubuntu"
```

platform_family

description	The entity's operating system family.
required	false
type	string
example	`"platform_family": "debian"`

platform_version

description	The entity's operating system version.
required	false
type	string
example	`"platform_version": "16.04"`

network

description	The entity's network interface list. See the network attributes for more information.
required	false
type	map

example


```
{
  "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

The entity's system architecture. This value is determined by the Go binary architecture, as a function of runtime.GOARCH. An `amd` system running a `386` binary will report the arch as `386`.

required

false

type

string

example


```
"arch": "amd64"
```

Network attributes

network_interface	
description	The list of network interfaces available on the entity, with their associated MAC and IP addresses.
required	false
type	array NetworkInterface

example	{
	"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	The network interface name.
required	false
type	string
example	`"name": "eth0"`

mac

description	The network interface's MAC address.
required	false
type	string
example	`"mac": "52:54:00:20:1b:3c"`

addresses

description	The list of IP addresses for the interface.
required	false
type	array
example	`"addresses": ["93.184.216.34/24", "2606:2800:220:1:248:1893:25c8:1946/10"]`

Deregistration attributes

handler	
description	The name of the handler to be called when an entity is deregistered.
required	false
type	string
example	`"handler": "email-handler"`

Examples

Entity definition


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

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Events

How do events work?

An event is a generic container used by Sensu to provide context to checksand/or metrics. The context, called “event data,” contains information about theoriginating entity and the corresponding check/metric result. An event mustcontain a check or metrics, and in certain cases, an event can contain both.These generic containers allow Sensu to handle different types of events in thepipeline. Since events are polymorphic in nature, it is important to neverassume their contents, or lack-thereof.

Check-only events

A Sensu event is created every time a check result is processed by the Sensuserver, regardless of the status indicated by the check result. An event iscreated by the agent on receipt of the check execution result. The agent willexecute any configured hooks the check might have. From there, it is forwarded to the Sensu backend for processing. Potentially noteworthy events maybe processed by one or more event handlers to do things such as send an email orinvoke an automated action.

Metric-only events

Sensu events can also be created when the agent receives metrics through the Statsd listener. The agent will translate the statsd metrics to SensuMetric Format, and place them inside an event. These events, since they do notcontain checks, bypass the store, and are sent off to the event pipeline and corresponding event handlers.

Check and metric events

Events that contain *both* a check and metrics, most likely originated from check output metric extraction. If a check is configured for metricextraction, the agent will parse the check output and transform it to SensuMetric 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.

Creating 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.

- Creating events using service checks
- Creating events using metric checks
- Creating events using the agent API
- Creating events using the agent TCP and UDP sockets
- Creating events using the StatsD listener

Creating 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.

Managing events

You can manage event using the Sensu dashboard, events API, and the sensuctl command line tool.

Viewing events

To list all events:

```
sensuctl event list
```

To show event details in the default output format:

```
sensuctl event info entity-name check-name
```

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

```
sensuctl event info entity-name check-name --format yaml
```

Deleting events

To delete an event:

```
sensuctl event delete entity-name check-name
```

You can use the `--skip-confirm` flag to skip the confirmation step.

```
sensuctl event delete entity-name check-name --skip-confirm
```

You should see a confirmation message on success.

```
Deleted
```

Resolving events

You can use sensuctl to change the status of an event to `0` (OK).Events resolved by sensuctl include the output message: “Resolved manually by sensuctl”.

```
sensuctl event resolve entity-name check-name
```

You should see a confirmation message on success.

```
Resolved
```

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

Using event data

Event data is powerful tool for automating monitoring workflows.For example, see the guide to reducing alert fatigue by filtering events based on the event `occurrences` attribute.

Events specification

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Events should always be of type `Event`.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "Event"`

api_version

description	Top-level attribute specifying the Sensu API group and version. For events in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

description	Top-level scope containing the event `namespace`. The `metadata` map is always at the top level of the check definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. See the metadata attributes reference for details.
required	Required for events in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	`"metadata": { "namespace": "default" }`

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
example	


```
"spec": {
  "check": {
    "check_hooks": null,
    "command": "/opt/sensu-plugins-
ruby/embedded/bin/metrics-curl.rb -u
\"http://localhost\"",
    "duration": 0.060790838,
    "env_vars": null,
    "executed": 1552506033,
    "handlers": [],
    "high_flap_threshold": 0,
    "history": [
      {
        "executed": 1552505833,
        "status": 0
      },
      {
        "executed": 1552505843,
        "status": 0
      }
    ],
    "interval": 10,
    "issued": 1552506033,
    "last_ok": 1552506033,
    "low_flap_threshold": 0,
    "metadata": {
      "name": "curl_timings",
      "namespace": "default"
    },
    "occurrences": 1,
    "occurrences_watermark": 1,
    "output": "sensu-go-sandbox.curl_timings.time_total
0.005 1552506033\nsensu-go-
sandbox.curl_timings.time_namelookup 0.004",
    "output_metric_format": "graphite_plaintext",
```

```
      "output_metric_handlers": [
        "influx-db"
      ],
      "proxy_entity_name": "",
      "publish": true,
      "round_robin": false,
      "runtime_assets": [],
      "state": "passing",
      "status": 0,
      "stdin": false,
      "subdue": null,
      "subscriptions": [
        "entity:sensu-go-sandbox"
      ],
      "timeout": 0,
      "total_state_change": 0,
      "ttl": 0
    },
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1552495139,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "subscriptions": [
        "entity:sensu-go-sandbox"
```

```
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-go-sandbox",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::5a94:f67a:1bfc:a579/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          }
        ]
      },
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804"
    },
    "user": "agent"
  },
  "metrics": {
    "handlers": [
      "influx-db"
    ],
    "points": [
      {
        "name": "sensu-go-
sandbox.curl_timings.time_total",
        "tags": [],
        "timestamp": 1552506033,
```

```
          "value": 0.005
        },
        {
          "name": "sensu-go-
sandbox.curl_timings.time_namelookup",
          "tags": [],
          "timestamp": 1552506033,
          "value": 0.004
        }
      ]
    },
    "timestamp": 1552506033
}
```


Metadata attributes

namespace	
description	The Sensu RBAC namespace that this event belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

Spec attributes

timestamp

description	Time that the event occurred in seconds since the Unix epoch
required	false
type	Integer
default	`0`
example	`"timestamp": 1522099512`

entity

description	The entity attributes from the originating entity (agent or proxy).
type	Map
required	true

example

```
"entity": {
  "deregister": false,
  "deregistration": {},
  "entity_class": "agent",
  "last_seen": 1552495139,
  "metadata": {
    "name": "sensu-go-sandbox",
    "namespace": "default"
  },
  "redact": [
    "password",
    "passwd",
    "pass",
    "api_key",
    "api_token",
    "access_key",
    "secret_key",
    "private_key",
```

```
      "secret"
    ],
    "subscriptions": [
      "entity:sensu-go-sandbox"
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-go-sandbox",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::5a94:f67a:1bfc:a579/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          }
        ]
      },
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.5.1804"
    },
    "user": "agent"
  }
```


check

description — The check definition used to create the event and information about the

status and history of the event. The check scope includes attributes described in the event specification and the check specification.

type	Map
required	true
example	


```
"check": {
  "check_hooks": null,
  "command": "/opt/sensu-plugins-
ruby/embedded/bin/metrics-curl.rb -u
\"http://localhost\"",
  "duration": 0.060790838,
  "env_vars": null,
  "executed": 1552506033,
  "handlers": [],
  "high_flap_threshold": 0,
  "history": [
    {
      "executed": 1552505833,
      "status": 0
    },
    {
      "executed": 1552505843,
      "status": 0
    }
  ],
  "interval": 10,
  "issued": 1552506033,
  "last_ok": 1552506033,
  "low_flap_threshold": 0,
  "metadata": {
    "name": "curl_timings",
    "namespace": "default"
  },
  "occurrences": 1,
  "occurrences_watermark": 1,
  "output": "sensu-go-sandbox.curl_timings.time_total
0.005",
  "output_metric_format": "graphite_plaintext",
```

```
  "output_metric_handlers": [
    "influx-db"
  ],
  "proxy_entity_name": "",
  "publish": true,
  "round_robin": false,
  "runtime_assets": [],
  "state": "passing",
  "status": 0,
  "stdin": false,
  "subdue": null,
  "subscriptions": [
    "entity:sensu-go-sandbox"
  ],
  "timeout": 0,
  "total_state_change": 0,
  "ttl": 0
}
```


metrics

description	The metrics collected by the entity in Sensu metric format. See the metrics attributes.
type	Map
required	false

example


```
"metrics": {
  "handlers": [
    "influx-db"
  ],
  "points": [
    {
      "name": "sensu-go-
sandbox.curl_timings.time_total",
      "tags": [],
```

```
        "timestamp": 1552506033,
        "value": 0.005
      },
      {
        "name": "sensu-go-
sandbox.curl_timings.time_namelookup",
        "tags": [],
        "timestamp": 1552506033,
        "value": 0.004
      }
    ]
}
```

Check attributes

Sensu events include a `check` scope containing information about how the event was created, including any attributes defined in the check specification, and information about the event and its history, including the attributes defined below.

duration	
description	Command execution time in seconds
required	false
type	Float
example	`"duration": 1.903135228`

executed	
description	Time that the check request was executed

required	false
type	Integer
example	`"executed": 1522100915`

history

description	Check status history for the last 21 check executions. See the history attributes.
required	false
type	Array
example	`"history": [ { "executed": 1552505983, "status": 0 }, { "executed": 1552505993, "status": 0 } ]`

issued

description	Time that the check request was issued in seconds since the Unix epoch
required	false
type	Integer

example	`"issued": 1552506033`

last_ok

description	The last time that the check returned an OK `status` (`0`) in seconds since the Unix epoch
required	false
type	Integer
example	`"last_ok": 1552506033`

occurrences

description	The number of times an event with the same status has occurred for the given entity and check
required	false
type	Integer
example	`"occurrences": 1`

occurrences_watermark

description	The highest number of occurrences for the given entity and check at the current status

required	false
type	Integer
example	`"occurrences_watermark": 1`

output

description	The output from the execution of the check command
required	false
type	String
example	`"output": "sensu-go-sandbox.curl_timings.time_total 0.005"`

state

description	The state of the check: `passing` (status `0`), `failing` (status other than `0`), or `flapping`. You can use the `low_flap_threshold` and `high_flap_threshold` check attributes to configure `flapping` state detection.
required	false
type	String
example	`"state": "passing"`

status	
description	Exit status code produced by the check ☐ `0` indicates “OK” ☐ `1` indicates “WARNING” ☐ `2` indicates “CRITICAL” ☐ exit status codes other than `0`, `1`, or `2` indicate an “UNKNOWN” or custom status
required	false
type	Integer
example	`"status": 0`

total_state_change	
description	The total state change percentage for the check’s history
required	false
type	Integer
example	`"total_state_change": 0`

History attributes

executed	
description	Time that the check request was executed in seconds since the Unix epoch

required	false
type	Integer
example	`"executed": 1522100915`

status

description	Exit status code produced by the check - `0` indicates “OK” - `1` indicates “WARNING” - `2` indicates “CRITICAL” - exit status codes other than `0`, `1`, or `2` indicate an “UNKNOWN” or custom status
required	false
type	Integer
example	`"status": 0`

Metric attributes

handlers

description	An array of Sensu handlers to use for events created by the check. Each array item must be a string.
required	false
type	Array
example	

example	"handlers": ["influx-db"]

points	
description	Metric data points including a name, timestamp, value, and tags. See the points attributes.
required	false
type	Array

example	"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
	  }
	]

Points attributes

name

description	The metric name in the format `$entity.$check.$metric` where `$entity` is the entity name, `$check` is the check name, and `$metric` is the metric name.
required	false
type	String
example	`"name": "sensu-go-sandbox.curl_timings.time_total"`

tags

description	Optional tags to include with the metric
required	false
type	Array
example	`"tags": []`

timestamp

description	Time that the metric was collected in seconds since the Unix epoch
required	false
type	Integer
example	`"timestamp": 1552506033`

value	
description	The metric value
required	false
type	Float
example	`"value": 0.005`

Examples

Example check-only event data


```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "check-cpu.sh -w 75 -c 90",
      "duration": 1.07055808,
      "env_vars": null,
      "executed": 1552594757,
      "handlers": [],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1552594757,
          "status": 0
        }
```

```
    ],
    "interval": 60,
    "issued": 1552594757,
    "last_ok": 1552594758,
    "low_flap_threshold": 0,
    "metadata": {
      "name": "check-cpu",
      "namespace": "default"
    },
    "occurrences": 1,
    "occurrences_watermark": 1,
    "output": "CPU OK - Usage:3.96\n",
    "output_metric_format": "",
    "output_metric_handlers": [],
    "proxy_entity_name": "",
    "publish": true,
    "round_robin": false,
    "runtime_assets": [],
    "state": "passing",
    "status": 0,
    "stdin": false,
    "subdue": null,
    "subscriptions": [
      "linux"
    ],
    "timeout": 0,
    "total_state_change": 0,
    "ttl": 0
  },
  "entity": {
    "deregister": false,
    "deregistration": {},
    "entity_class": "agent",
    "last_seen": 1552594641,
    "metadata": {
      "name": "sensu-centos",
      "namespace": "default"
    },
    "redact": [
      "password",
```

```
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "subscriptions": [
        "linux",
        "entity:sensu-centos"
      ],
      "system": {
        "arch": "amd64",
        "hostname": "sensu-centos",
        "network": {
          "interfaces": [
            {
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ],
              "name": "lo"
            },
            {
              "addresses": [
                "10.0.2.15/24",
                "fe80::9688:67ca:3d78:ced9/64"
              ],
              "mac": "08:00:27:11:ad:d2",
              "name": "enp0s3"
            },
            {
              "addresses": [
                "172.28.128.3/24",
                "fe80::a00:27ff:fe6b:c1e9/64"
              ],
              "mac": "08:00:27:6b:c1:e9",
              "name": "enp0s8"
```

```
            }
          ]
        },
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.4.1708"
      },
      "user": "agent"
    },
    "timestamp": 1552594758
  }
}
```


Example event with check and metric data


```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
  "spec": {
    "check": {
      "check_hooks": null,
      "command": "/opt/sensu-plugins-ruby/embedded/bin/metrics-curl.rb -u
\"http://localhost\"",
      "duration": 0.060790838,
      "env_vars": null,
      "executed": 1552506033,
      "handlers": [],
      "high_flap_threshold": 0,
      "history": [
        {
          "executed": 1552505833,
          "status": 0
        },
        {
```

```
            "executed": 1552505843,
            "status": 0
          }
        ],
        "interval": 10,
        "issued": 1552506033,
        "last_ok": 1552506033,
        "low_flap_threshold": 0,
        "metadata": {
          "name": "curl_timings",
          "namespace": "default"
        },
        "occurrences": 1,
        "occurrences_watermark": 1,
        "output": "sensu-go-sandbox.curl_timings.time_total 0.005
1552506033\nsensu-go-sandbox.curl_timings.time_namelookup 0.004",
        "output_metric_format": "graphite_plaintext",
        "output_metric_handlers": [
          "influx-db"
        ],
        "proxy_entity_name": "",
        "publish": true,
        "round_robin": false,
        "runtime_assets": [],
        "state": "passing",
        "status": 0,
        "stdin": false,
        "subdue": null,
        "subscriptions": [
          "entity:sensu-go-sandbox"
        ],
        "timeout": 0,
        "total_state_change": 0,
        "ttl": 0
      },
      "entity": {
        "deregister": false,
        "deregistration": {},
        "entity_class": "agent",
        "last_seen": 1552495139,
```

```
    "metadata": {
      "name": "sensu-go-sandbox",
      "namespace": "default"
    },
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ],
    "subscriptions": [
      "entity:sensu-go-sandbox"
    ],
    "system": {
      "arch": "amd64",
      "hostname": "sensu-go-sandbox",
      "network": {
        "interfaces": [
          {
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ],
            "name": "lo"
          },
          {
            "addresses": [
              "10.0.2.15/24",
              "fe80::5a94:f67a:1bfc:a579/64"
            ],
            "mac": "08:00:27:8b:c9:3f",
            "name": "eth0"
          }
        ]
      },
```

```
        "os": "linux",
        "platform": "centos",
        "platform_family": "rhel",
        "platform_version": "7.5.1804"
      },
      "user": "agent"
    },
    "metrics": {
      "handlers": [
        "influx-db"
      ],
      "points": [
        {
          "name": "sensu-go-sandbox.curl_timings.time_total",
          "tags": [],
          "timestamp": 1552506033,
          "value": 0.005
        },
        {
          "name": "sensu-go-sandbox.curl_timings.time_namelookup",
          "tags": [],
          "timestamp": 1552506033,
          "value": 0.004
        }
      ]
    },
    "timestamp": 1552506033
  }
}
```


Example metric-only event


```
{
  "type": "Event",
  "api_version": "core/v2",
  "metadata": {
    "namespace": "default"
  },
```

```
  "spec": {
    "entity": {
      "deregister": false,
      "deregistration": {},
      "entity_class": "agent",
      "last_seen": 1552495139,
      "metadata": {
        "name": "sensu-go-sandbox",
        "namespace": "default"
      },
      "redact": [
        "password",
        "passwd",
        "pass",
        "api_key",
        "api_token",
        "access_key",
        "secret_key",
        "private_key",
        "secret"
      ],
      "subscriptions": [
        "entity:sensu-go-sandbox"
      ],
      "system": {
        "arch": "amd64",
        "hostname": "sensu-go-sandbox",
        "network": {
          "interfaces": [
            {
              "addresses": [
                "127.0.0.1/8",
                "::1/128"
              ],
              "name": "lo"
            },
            {
              "addresses": [
                "10.0.2.15/24",
                "fe80::5a94:f67a:1bfc:a579/64"
```

```
                ],
                "mac": "08:00:27:8b:c9:3f",
                "name": "eth0"
              }
            ]
          },
          "os": "linux",
          "platform": "centos",
          "platform_family": "rhel",
          "platform_version": "7.5.1804"
        },
        "user": "agent"
      },
      "metrics": {
        "handlers": [
          "influx-db"
        ],
        "points": [
          {
            "name": "sensu-go-sandbox.curl_timings.time_total",
            "tags": [],
            "timestamp": 1552506033,
            "value": 0.005
          },
          {
            "name": "sensu-go-sandbox.curl_timings.time_namelookup",
            "tags": [],
            "timestamp": 1552506033,
            "value": 0.004
          }
        ]
      },
      "timestamp": 1552506033
    }
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Filters

How do Sensu filters work?

Sensu filters are applied when **event handlers** are configured to use one ormore filters. Prior to executing a handler, the Sensu server will apply anyfilters configured for the handler to the **event** data. If the event is notremoved by the filter(s), the handler will be executed. Thefilter analysis flow performs these steps:

- When the Sensu server is processing an event, it will check for the definitionof a `handler` (or `handlers`). Prior to executing each handler, the Sensuserver will first apply any configured `filters` for the handler.
- If multiple `filters` are configured for a handler, they are executedsequentially.
- Filter `expressions` are compared with event data.
- 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 furtheranalysis is performed and the event handler will not be executed.

NOTE: Filters specified in a **handler set** *definition have no effect. Filters mustbe specified in individual handler definitions.*

Inclusive and exclusive filtering

Filters can be *inclusive* `"action": "allow"` (replaces `"negate": false` inSensu 1) or *exclusive* `"action": "deny"` (replaces `"negate": true` in Sensu1). Configuring a handler to use multiple *inclusive* filters is the equivalentof using an `AND` query operator (only handle events if they match *inclusive* filter `x AND y AND z`). Configuring a handler to use multiple *exclusive* filters is the equivalent of using an `OR` operator (onlyhandle events if they don't match `x OR y OR z`).

- **Inclusive filtering**: by setting the filter definition attribute `"action":"allow"`, only events that match the defined filter expressions are handled.
- **Exclusive filtering**: by setting the filter definition attribute `"action":"deny"`, events are only handled if they do not match the defined filterexpressions.

Filter expression comparison

Filter expressions are compared directly with their event data counterparts. Forinclusive filter definitions (like `"action": "allow"`), matching expressionswill result in the filter returning a `true` value; for exclusive filterdefinitions (like `"action": "deny"`), matching expressions will result in the filter returning a `false` value, and the event will not pass through thefilter. Filters that return a true value will continue to be processed viaadditional filters (if defined), mutators (if defined), and handlers.

Filter expression evaluation

When more complex conditional logic is needed than direct filter expressioncomparison, Sensu filters provide support for expression evaluation using Otto. Otto is an ECMAScript 5 (JavaScript) VM,and evaluates javascript expressions that are provided in the filter.There are some caveats to using Otto; most notably, the regular expressionsspecified in ECMAScript 5 do not all work. See the Otto README for more details.

Filter assets

Sensu filters can have assets that are included in their execution context.When valid assets are

associated with a filter, Sensu evaluates anyfiles it finds that have a ".js" extension before executing a filter. Theresult of evaluating the scripts is cached for a given asset set, for thesake of performance.

Built-in filters

Sensu includes built-in filters to help you customize event pipelines for metrics and alerts.To start using built-in filters, see the guides to sending Slack alerts and planning maintenances.

Built-in filter: only incidents

The incidents filter is included in every installation of the Sensu backend.You can use the incidents filter to allow only high priority events through a Sensu pipeline.For example, you can use the incidents filter to reduce noise when sending notifications to Slack.When applied to a handler, the incidents filter allows only warning (`"status": 1`), critical (`"status": 2`), and resolution events to be processed.

To use the incidents filter, include the `is_incident` filter in the handler configuration `filters` array:

```
{
  "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
    ],
    "filters": [
      "is_incident"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
```

```
    "type": "pipe"
  }
}
```

The `is_incident` filter applies the following filtering logic:

status	allow	discard
0		
1		
2		
other		
1 –> 0 or 2 –> 0 (resolution event)		

Built-in filter: allow silencing

Sensu silencing lets you suppress execution of event handlers on an on-demand basis, giving you the ability to quiet incoming alerts and plan maintenances.

To allow silencing for an event handler, add the `not_silenced` filter to the handler configuration `filters` array:

```
{
  "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
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
}
```

When applied to a handler configuration, the `not_silenced` filter silences events that include the `"silenced": true` attribute. The handler in the example above uses both the silencing and incidents filters, preventing low priority and silenced events from being sent to Slack.

Built-in filter: has metrics

The metrics filter is included in every installation of the Sensu backend.When applied to a handler, the metrics filter allows only events containing Sensu metrics to be processed.You can use the metrics filter to prevent handlers that require metrics from failing in case of an error in metric collection.

To use the metrics filter, include the `has_metrics` filter in the handler configuration `filters` array:

```
{
  "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` filter allows only events that include a `metrics` scope.

Building filter expressions

You can write custom filter expressions as Sensu query expressions using the event data attributes described in this section.For more information about event attributes, see the event reference.

Syntax quick reference

operator	description
`===` / `!==`	Identity operator / Nonidentity operator
`==` / `!=`	Equality operator / Inequality operator
`&&` / `||`	Logical AND / Logical OR
`<` / `>`	Less than / Greater than
`<=` / `>=`	Less than or equal to / Greater than or equal to

Event attributes available to filters

attribute	type	description
`event.has_check`	boolean	Returns true if the event contains check data
`event.has_metrics`	boolean	Returns true if the event contains metrics
`event.is_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	If the check is configured to discard check output from event data
`event.check.duration`	float	Command execution time in

		seconds
`event.check.env_vars`	array	Environment variables used with command execution
`event.check.executed`	integer	Time that the check was executed in seconds since the Unix epoch
`event.check.handlers`	array	Sensu event handlers assigned to the check
`event.check.high_flap_threshold`	integer	The check's flap detection high threshold in percent state change
`event.check.history`	array	Check status history for the last 21 check executions
`event.check.hooks`	array	Check hook execution data
`event.check.interval`	integer	The check execution frequency in seconds
`event.check.issued`	integer	Time that the check request was issued in seconds since the Unix epoch
`event.check.labels`	map	Custom labels applied to the check
`event.check.last_ok`	integer	The last time that the check returned an OK status (`0`) in seconds since the Unix epoch
`event.check.low_flap_threshold`	integer	The check's flap detection low threshold in percent state change
`event.check.max_output_size`	integer	Maximum size, in bytes, of stored check outputs
`event.check.name`	string	Check name
`event.check.occurrences`	integer	The number of times an event with the same status has occurred for

		the given entity and check
`event.check.occurrences_watermark`	integer	The highest number of occurrences for the given entity and check at the current status
`event.check.output`	string	The output from the execution of the check command
`event.check.output_metric_format`	string	The metric format generated by the check command: `nagios_perfdata`, `graphite_plaintext`, `influxdb_line`, or `opentsdb_line`
`event.check.output_metric_handlers`	array	Sensu metric handlers assigned to the check
`event.check.proxy_entity_name`	string	The entity name, used to create a proxy entity for an external resource
`event.check.proxy_requests`	map	Proxy request configuration
`event.check.publish`	boolean	If the check is scheduled automatically
`event.check.round_robin`	boolean	If the check is configured to be executed in a round-robin style
`event.check.runtime_assets`	array	Sensu assets used by the check
`event.check.state`	string	The state of the check: `passing` (status `0`), `failing` (status other than `0`), or `flapping`
`event.check.status`	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	If 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) in seconds until the event is considered stale
`event.metrics.handlers`	array	Sensu metric handlers assigned to the check
`event.metrics.points`	array	Metric data points including a name, timestamp, value, and tags

Entity attributes available to filters

attribute	type	description
`event.entity.annotations`	map	Custom annotations assigned to the entity
`event.entity.deregister`	boolean	If the agent entity should be removed when it stops sending keepalive messages
`event.entity.deregistration`	map	A map containing a handler name, for use when an entity is deregistered
`event.entity.entity_class`	string	The entity type: usually `agent` or `proxy`
`event.entity.labels`	map	Custom labels assigned to the entity

`event.entity.last_seen`	integer	Timestamp the entity was last seen, in seconds since the Unix epoch
`event.entity.name`	string	Entity name
`event.entity.redact`	array	List of items to redact from log messages
`event.entity.subscriptions`	array	List of subscriptions assigned to the entity
`event.entity.system`	map	Information about the entity's system
`event.entity.system.arch`	string	The entity's system architecture
`event.entity.system.hostname`	string	The entity's hostname
`event.entity.system.network`	map	The entity's network interface list
`event.entity.system.os`	string	The entity's operating system
`event.entity.system.platform`	string	The entity's operating system distribution
`event.entity.system.platform_family`	string	The entity's operating system family
`event.entity.system.platform_version`	string	The entity's operating system version
`event.entity.user`	string	Sensu RBAC username used by the agent entity

Filter specification

Top-level attributes

type

description	Top-level attribute specifying the `sensuctl create` resource type. Filters should always be of type `EventFilter`.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "EventFilter"`

api_version

description	Top-level attribute specifying the Sensu API group and version. For filters in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for filter definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the filter, including the `name` and `namespace` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the filter definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. See the metadata attributes reference 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	<pre>"metadata": {
	  "name": "filter-weekdays-only",
	  "namespace": "default",
	  "labels": {
	  "region": "us-west-1"
	  },
	  "annotations": {
	  "slack-channel" : "#monitoring"
	  }
	}</pre>

spec

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

example	<pre>"spec": {
	  "action": "allow",
	  "expressions": [
	  "event.entity.namespace == 'production'"
	  ],
	  "runtime_assets": []
	}</pre>

Spec attributes

action

description	Action to take with the event if the filter expressions match. *NOTE: see Inclusive and exclusive filtering for more information.*
required	true
type	String
allowed values	`allow`, `deny`
example	`"action": "allow"`

expressions

description	Filter expressions to be compared with event data. Note that event metadata can be referenced without including the `metadata` scope, for example: `event.entity.namespace`.
required	true
type	Array
example	`"expressions": [ "event.check.team == 'ops'" ]`

runtime_assets

description	Assets to be applied to the filter's execution context. JavaScript files in the lib directory of the asset will be evaluated.

required	false
type	Array of String
default	[]
example	`"runtime_assets": ["underscore"]`

Metadata attributes

name	
description	A unique string used to identify the filter. Filter names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each filter must have a unique name within its namespace.
required	true
type	String
example	`"name": "filter-weekdays-only"`

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

example	<pre>"namespace": "production"</pre>

labels

description	Custom attributes to include with event data, which can be queried like regular attributes.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example	<pre>"labels": {
	  "environment": "development",
	  "region": "us-west-2"
	}</pre>

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are *not* used internally by Sensu and cannot be used to identify filters. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`
example	

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

Filter Examples

Minimum required filter attributes

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "filter_minimum",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.check.occurrences == 1"
    ]
  }
}
```

Handling production events

The following filter allows only events with a custom entity label `"environment": "production"` to be handled.

```
{
  "type": "EventFilter",
```

```
  "api_version": "core/v2",
  "metadata": {
    "name": "production_filter",
    "namespace": "default"
  },
  "spec": {
    "action": "allow",
    "expressions": [
      "event.entity.labels.environment == 'production'"
    ]
  }
}
```

Handling non-production events

The following filter discards events with a custom entity label `"environment": "production"`, allowing only events without an `environment` label or events with `environment` set to something other than `production` to be handled.Note that `action` is `deny`, making this an exclusive filter; if evaluationreturns false, the event is handled.

```
{
  "type": "EventFilter",
  "api_version": "core/v2",
  "metadata": {
    "name": "not_production",
    "namespace": "default"
  },
  "spec": {
    "action": "deny",
    "expressions": [
      "event.entity.labels.environment == 'production'"
    ]
  }
}
```

Handling state change only

Some teams migrating to Sensu have asked about reproducing the behavior of theirold monitoring system which alerts only on state change. This `state_change_only` inclusive filter provides such.

```
{
  "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": []
  }
}
```

Handling repeated events

The following example filter definition, entitled `filter_interval_60_hourly`,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 moduloperator calculation(calculating the remainder after dividing `occurrences` by 60).

```
{
  "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": []
  }
}
```

The next example will apply the same logic as the previous example, but forchecks with a 30 second `interval`.

```
{
  "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": []
  }
}
```

Handling events during office hours only

This filter evaluates the event timestamp to determine if the event occurredbetween 9 AM and 5 PM UTC on a weekday. Remember that `action` is equal to `allow`, so this is an inclusive filter. If

evaluation returns false, the eventwill not be handled.

```
{
  "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": []
  }
}
```

Using JavaScript libraries with Sensu filters

You can include JavaScript libraries in their filter execution context withassets. For instance, assuming you've packaged underscore.js into a Sensuasset, you could then use functions from the underscore library for filterexpressions.

```
{
  "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"]
    }
  }
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Handlers

How do Sensu handlers work?

Handlers actions are executed by the Sensu backend on events, and there areseveral types of handlers available. The most common handler type is the `pipe` handler, which works very similarly to how checks work, enabling Sensu tointeract with almost any computer program via standard streams.

- **Pipe handlers**. Pipe handlers pipe event data into arbitrary commands via `STDIN`.
- **TCP/UDP handlers**. TCP and UDP handlers send event data to a remote socket.
- **Handler sets**. Handler sets (also called “set handlers”) are used to groupevent handlers, making it easy to manage groups of actions that should beexecuted for certain types of events.

Pipe handlers

Pipe handlers are external commands that can consume event data via STDIN.

Pipe handler command

Pipe handler definitions include a `command` attribute, which is a command to beexecuted by the Sensu backend.

Pipe handler command arguments

Pipe handler `command` attributes may include command line arguments forcontrolling the behavior of the `command` executable.

TCP/UDP handlers

TCP and UDP handlers enable Sensu to forward event data to arbitrary TCP or UDPsockets for external services to consume.

Handler sets

Handler set definitions allow groups of handlers (individual collectionsof actions to take on event data) to be referenced via a single named handlerset.

NOTE: Attributes defined on handler sets do not apply to the handlers theyinclude. For example, `filters`, and `mutator` attributes definedin a handler set will have no effect.

Handling keepalive events

Sensu keepalives are the heartbeat mechanism used to ensure that all registered Sensu agents are operational and able to 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.

Let's say 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:

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "keepalive",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack"
    ]
  }
}
```

Handler specification

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Handlers should always be of type `Handler`.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "Handler"`

api_version

description	Top-level attribute specifying the Sensu API group and version. For handlers in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for handler definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

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

```
"metadata": {
  "name": "handler-slack",
  "namespace": "default",
  "labels": {
    "region": "us-west-1"
  },
  "annotations": {
```

```
    "slack-channel" : "#monitoring"
  }
}
```

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

```
"spec": {
  "type": "tcp",
  "socket": {
    "host": "10.0.1.99",
    "port": 4444
  },
  "metadata" : {
    "name": "tcp_handler",
    "namespace": "default"
  }
}
```

Spec attributes

type	
description	The handler type.
required	true

type	String
allowed values	`pipe`, `tcp`, `udp` & `set`
example	`"type": "pipe"`

filters

description	An array of Sensu event filters (names) to use when filtering events for the handler. Each array item must be a string.
required	false
type	Array
example	`"filters": ["occurrences", "production"]`

mutator

description	The Sensu event mutator (name) to use to mutate event data for the handler.
required	false
type	String
example	`"mutator": "only_check_output"`

timeout

description	The handler execution duration timeout in seconds (hard stop). Only used by `pipe` and `tcp` handler types.
required	false
type	Integer
default	`60` (for `tcp` handler)
example	`"timeout": 30`

command

description	The handler command to be executed. The event data is passed to the process via `STDIN`.*NOTE: the `command` attribute is only supported for Pipe handlers (i.e. handlers configured with `"type": "pipe"`).*
required	true (if `type` equals `pipe`)
type	String
example	`"command": "/etc/sensu/plugins/pagerduty.go"`

env_vars

description	An array of environment variables to use with command execution.*NOTE: the `env_vars` attribute is only supported for Pipe handlers (i.e. handlers configured with `"type": "pipe"`).*
required	false
type	Array

example	<pre>"env_vars":
	["API_KEY=0428d6b8nb51an4d95nbe28nf90865a66af5"]</pre>

socket

description	The `socket` definition scope, used to configure the TCP/UDP handler socket. *NOTE: the `socket` attribute is only supported for TCP/UDP handlers (i.e. handlers configured with `"type": "tcp"` or `"type": "udp"`).*
required	true (if `type` equals `tcp` or `udp`)
type	Hash
example	<pre>"socket": {}</pre>

handlers

description	An array of Sensu event handlers (names) to use for events using the handler set. Each array item must be a string. *NOTE: the `handlers` attribute is only supported for handler sets (i.e. handlers configured with `"type": "set"`).*
required	true (if `type` equals `set`)
type	Array
example	<pre>"handlers": ["pagerduty", "email", "ec2"]</pre>

runtime_assets

description	An array of Sensu assets (names), required at runtime for the execution of the `command`
required	false
type	Array
example	`"runtime_assets": ["ruby-2.5.0"]`

Metadata attributes

name	
description	A 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	`"name": "handler-slack"`

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

default	

example

```
"namespace": "production"
```

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize handlers into meaningful collections that can be selected using filters and tokens.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.
default	`null`

example

```
"labels": {
  "environment": "development",
  "region": "us-west-2"
}
```

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are *not* used internally by Sensu and cannot be used to identify handlers. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.

default	`null`

example

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

`socket` attributes

host

description	The socket host address (IP or hostname) to connect to.
required	true
type	String

example

```
"host": "8.8.8.8"
```

port

description	The socket port to connect to.
required	true
type	Integer

example

```
"port": 4242
```

Handler examples

Minimum required pipe handler attributes

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "pipe_handler_minimum",
    "namespace": "default"
  },
  "spec": {
    "command": "command-example",
    "type": "pipe"
  }
}
```

Minimum required TCP/UDP handler attributes

This is an example of a `tcp` type handler. Changing the type from `tcp` to `udp` gives you the minimum configuration for a `udp` type handler.

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata": {
    "name": "tcp_udp_handler_minimum",
    "namespace": "default"
  },
  "spec": {
    "type": "tcp",
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
```

```
      }
    }
  }
```

Sending slack alerts

This handler will send alerts to a channel named `monitoring` with theconfigured webhook URL, using the `handler-slack` executable command.

```
{
  "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
    ],
    "filters": [
      "is_incident",
      "not_silenced"
    ],
    "handlers": [],
    "runtime_assets": [],
    "timeout": 0,
    "type": "pipe"
  }
}
```

Sending event data to a TCP socket

This handler will forward event data to a TCP socket (10.0.1.99:4444) andwill timeout if an acknowledgement (`ACK`) is not received within 30 seconds.

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "tcp_handler",
    "namespace": "default"
  },
  "spec": {
    "type": "tcp",
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
}
```

Sending event data to a UDP socket

The following example will also forward event data but to UDP socket instead(ex: 10.0.1.99:4444).

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "udp_handler",
    "namespace": "default"
  },
  "spec": {
    "type": "udp",
    "socket": {
      "host": "10.0.1.99",
      "port": 4444
    }
  }
```

```
}
```

Executing multiple handlers

The following example handler will execute three handlers: `slack`, `tcp_handler`, and `udp_handler`.

```
{
  "type": "Handler",
  "api_version": "core/v2",
  "metadata" : {
    "name": "notify_all_the_things",
    "namespace": "default"
  },
  "spec": {
    "type": "set",
    "handlers": [
      "slack",
      "tcp_handler",
      "udp_handler"
    ]
  }
}
```

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Hooks

How do hooks work?

Hooks are executed in response to the result of a check command executionand based on the exit status code of that command (ex: `1`).Hook commands can optionally receive JSON serialized Sensu client data viaSTDIN.You can create, manage, and reuse hooks independently of checks.

Check response types

Each **type** of response (ex: `non-zero`) can contain one or more hooks, andcorrespond to one or more exit status code. Hooks are executed, in order ofprecedence, based on their type:

1. `1` to `255`
2. `ok`
3. `warning`
4. `critical`
5. `unknown`
6. `non-zero`

You can assign one or more hooks to a check in the check definition.See the check specification to configure the `check_hooks` attribute.

Check hooks

The hook command output, status, executed timestamp and duration are capturedand published in the resulting event.

Hooks specification

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Hooks should always be of type `HookConfig`.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "HookConfig"`

api_version	
description	Top-level attribute specifying the Sensu API group and version. For hooks in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the hook, including the `name` and `namespace` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the hook definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. See the metadata attributes reference for details.
required	Required for hook definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	`"metadata": { "name": "process_tree", "namespace": "default", "labels": { "region": "us-west-1" }, "annotations": { "slack-channel" : "#monitoring" } }`

spec

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

```
  "command": "ps aux",
  "timeout": 60,
  "stdin": false
}
```

Spec attributes

command

description	The hook command to be executed.
required	true
type	String
example	`"command": "sudo /etc/init.d/nginx start"`

timeout

description	The hook execution duration timeout in seconds (hard stop).
required	false
type	Integer
default	60
example	`"timeout": 30`

stdin

description	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. This attribute cannot be used with existing Sensu check plugins, nor Nagios plugins etc, as Sensu agent will wait indefinitely for the hook process to read and close STDIN.
required	false
type	Boolean
default	false
example	`"stdin": true`

Metadata attributes

name	
description	A unique string used to identify the hook. Hook names cannot contain special characters or spaces (validated with Go regex `\A[\w\.\-]+\z`). Each hook must have a unique name within its namespace.
required	true
type	String
example	`"name": "process_tree"`

namespace	
description	The Sensu RBAC namespace that this hook belongs to.

required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize hooks into meaningful collections that can be selected using filters and tokens.
required	false
type	Map of key-value pairs. Keys can contain only letters, numbers, and underscores, but must start with a letter. Values can be any valid UTF-8 string.
default	`null`
example	`"labels": { "environment": "development", "region": "us-west-2" }`

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are *not* used internally by Sensu and cannot be used to identify hooks. You can use annotations to add data that helps people or external tools interacting with Sensu.

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

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

Examples

Rudimentary auto-remediation

Hooks can be used for rudimentary auto-remediation tasks, for example, startinga process that is no longer running.

NOTE: Using hooks for auto-remediation should be approachedcarefully, as they run without regard to the number of eventoccurrences.

```
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "restart_nginx",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "command": "sudo systemctl start nginx",
    "timeout": 60,
```

```
    "stdin": false
  }
}
```

Capture the process tree

Hooks can also be used for automated data gathering for incident triage, forexample, a check hook could be used to capture the process tree when a processhas been determined to be not running etc.

```
{
  "type": "HookConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "process_tree",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "command": "ps aux",
    "timeout": 60,
    "stdin": false
  }
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

License management

Enterprise features for Sensu Go are available in version 5.2.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest documentation to manage your enterprise license.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Mutators

How do mutators work?

A handler can specify a mutator to transform event data. Mutators are executedprior to the execution of a handler. If the mutator executes successfully, the modified eventdata is returned to the handler, and the handler is then executed. If the mutatorfails to execute, an error will be logged, and the handler will not be executed.

- When Sensu server processes an event, it will check the handler for thepresence of a mutator, and execute that mutator before executing the handler.
- If the mutator executes successfully (it returns an exit status code of 0), modifiedevent data is provided to the handler, and the handler is executed.
- If the mutator fails to execute (it returns a non-zero exit status code, orfails to complete within its configured timeout), an error will be logged andthe handler will not execute.

Mutator specification

- Accepts input/data via `STDIN`
- Able to parse JSON event data
- Outputs JSON data (modified event data) to `STDOUT` or `STDERR`
- Produces an exit status code to indicate state:
  - `0` indicates OK status

- exit codes other than `0` indicate failure

Commands

Each Sensu mutator definition defines a command to be executed. Mutator commands are executable commands which will be executed on a Sensu server, run as the `sensu user`. Most mutator commands are provided by Sensu Plugins.

Sensu mutator `command` attributes may include command line arguments forcontrolling the behavior of the `command` executable. Many Sensu mutator pluginsprovide support for command line arguments for reusability.

How and where are mutator commands executed?

As mentioned above, all mutator commands are executed by a Sensu server as the `sensu` user. Commands must be executable files that are discoverable on the Sensu server system (installed in a system `$PATH` directory).

NOTE: By default, the 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 it can be accepted by Graphite.

To use the only check output mutator, include the `only_check_output` mutator in the handler configuration `mutator` string:

```
{
  "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"
  }
}
```

Mutators specification

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Mutators should always be of type `Mutator`.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "Mutator"`

api_version

description	Top-level attribute specifying the Sensu API group and version. For mutators in Sensu backend version 5.0, this attribute should always be `core/v2`.
required	Required for mutator definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

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

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

```
}
```

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

example	<pre>"spec": {
	  "command": "example_mutator.go",
	  "timeout": 0,
	  "env_vars": [],
	  "runtime_assets": []
	}</pre>

Spec attributes

command	
description	The mutator command to be executed by Sensu server.
required	true
type	String
example	<pre>"command": "/etc/sensu/plugins/mutated.go"</pre>

env_vars

description	An array of environment variables to use with command execution.
required	false
type	Array
example	`"env_vars": ["RUBY_VERSION=2.5.0"]`

timeout

description	The mutator execution duration timeout in seconds (hard stop).
required	false
type	integer
example	`"timeout": 30`

runtime_assets

description	An array of Sensu assets (names), required at runtime for the execution of the `command`
required	false
type	Array
example	`"runtime_assets": ["ruby-2.5.0"]`

Metadata attributes

name

description	A 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	`"name": "example-mutator"`

namespace

description	The Sensu RBAC namespace that this mutator belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels

description	Custom attributes to include with event data, which can be queried like regular attributes. You can use labels to organize mutators into meaningful collections that can be selected using filters and tokens.

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

example	<pre>"labels": {
	  "environment": "development",
	  "region": "us-west-2"
	}</pre>

annotations

description	Arbitrary, non-identifying metadata to include with event data. In contrast to labels, annotations are not used internally by Sensu and cannot be used to identify mutators. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example	<pre>"annotations": {
	  "managed-by": "ops",
	  "slack-channel": "#monitoring",
	  "playbook": "www.example.url"
	}</pre>

Examples

The following Sensu mutator definition uses an imaginary Sensu plugin called `example_mutator.go` to modify event data prior to handling the event.

Mutator definition

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "example-mutator",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "command": "example_mutator.go",
    "timeout": 0,
    "env_vars": [],
    "runtime_assets": []
  }
}
```

Minimum required mutator attributes

```
{
  "type": "Mutator",
  "api_version": "core/v2",
  "metadata": {
    "name": "mutator_minimum",
    "namespace": "default"
  },
  "spec": {
    "command": "example_mutator.go"
  }
```

```
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Role-based access control

Sensu role-based access control (RBAC) helps different teams and projects share a Sensu instance. RBAC allows management and access of users and resources based on **namespaces**, **groups**, **roles**, and **bindings**.

- **Namespaces** partition resources within Sensu. Sensu entities, checks, handlers, and other namespaced resources belong to a single namespace.
- **Roles** create sets of permissions (get, delete, etc.) tied to resource types. **Cluster roles** apply permissions across namespaces and include access to cluster-wide resources like users and namespaces.
- **Users** represent a person or agent that interacts with Sensu. Users can belong to one or more **groups**.
- **Role bindings** assign a role to a set of users and groups within a namespace; **cluster role bindings** assign a cluster role to a set of users and groups cluster-wide.

Sensu access controls apply to sensuctl, the Sensu API, and the Sensu dashboard.

Namespaces

Namespaces help teams use different resources (entities, checks, handlers, etc.) 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 need to be unique within a namespace, but not across namespaces.

To create and manage namespaces, configure sensuctl as the default `admin` user or create a cluster role with `namespaces` permissions.

Default namespace

Every Sensu backend includes a `default` namespace.All resources created without a specified namespace are created within the `default` namespace.

Viewing namespaces

You can use sensuctl to view all namespaces within Sensu:

```
sensuctl namespace list
```

Creating a namespace

You can use sensuctl to create a namespace.For example, the following command creates a namespace called `production`:

```
sensuctl namespace create production
```

Namespace names can contain alphanumeric characters and hyphens, but must begin and end with an alphanumeric character.

Managing namespaces

You can use sensuctl to view, create, and delete namespaces.

To delete a namespace:

```
sensuctl namespace delete [NAMESPACE-NAME]
```

To get help managing namespaces with sensuctl:

```
sensuctl namespace help
```

Assigning a resource to a namespace

You can assign a resource to a namespace in the resource definition.Only resources belonging 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:

```
{
  "type": "CheckConfig",
  "api_version": "core/v2",
  "metadata": {
    "name": "check-cpu",
    "namespace": "default"
  },
  "spec": {
    "check_hooks": null,
    "command": "check-cpu.sh -w 75 -c 90",
    "handlers": ["slack"],
    "interval": 30,
    "subscriptions": ["system"],
    "timeout": 0,
    "ttl": 0
  }
}
```

See the reference docs for the corresponding resource type to create resource definitions.

Namespace specification

Attributes

name	
description	The name of the namespace. Names can contain alphanumeric characters and hyphens, but must begin and end with an alphanumeric character.
required	true
type	String
example	`"name": "production"`

Namespace example

The following example is in `wrapped-json` format for use with `sensuctl create`.

```
{
  "type": "Namespace",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "name": "default"
  }
}
```

Resources

Permissions within Sensu are scoped to resource types, like checks, handlers, and users.You can use resource types to configure permissions in Sensu roles and cluster roles.

Namespaced resource types

Namespaced resources must belong to a single namespace and can be accessed by roles and cluster roles.

Type	Description
`assets`	Asset resources within a namespace
`checks`	Check resources within a namespace
`entities`	Entity resources within a namespace
`events`	Event resources within a namespace
`extensions`	Placeholder type
`filters`	Filter resources within a namespace
`handlers`	Handler resources within a namespace
`hooks`	Hook resources within a namespace
`mutators`	Mutator resources within a namespace
`rolebindings`	Namespace-specific role assigners
`roles`	Namespace-specific permission sets
`silenced`	Silencing resources within a namespace

Cluster-wide resource types

Cluster-wide resources cannot be assigned to a namespace and can only be accessed by cluster roles.

Type	Description
`cluster`	Sensu clusters running multiple Sensu backends
`clusterrolebindings`	Cluster-wide role assigners
`clusterroles`	Cluster-wide permission sets
`namespaces`	Resource partitions within a Sensu instance
`users`	People or agents interacting with Sensu

Special resource types

Special resources types can be accessed 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 wish to deny a certain type, you can't use the `*` type and 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 which interacts with Sensu.Users and groups can be assigned one or more roles and inherit all permissions from each role assigned to them.

You can use your Sensu username and password to configure sensuctl or log in to the dashboard.

Default user

By default, Sensu includes a global `admin` user that you can use to manage Sensu and create new

users.

attribute	value
username	`admin`
password	`P@ssw0rd!`
groups	`cluster-admins`
cluster role	`cluster-admin`
cluster role binding	`cluster-admin`

We **strongly** recommended changing the default password for the admin user immediately.Once authenticated, you can change the password using the `change-password` command.

```
sensuctl user change-password
```

Sensu also includes an `agent` user that is used internally by the Sensu agent.You can configure an agent's user credentials using the `user` and `password` agent configuration flags.

Viewing users

You can use sensuctl to see a list of all users within Sensu.The following example returns a list of users in `yaml` format for use with `sensuctl create`.

```
sensuctl user list --format yaml
```

Creating a user

You can use sensuctl to create a user.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
```

Assigning user permissions

To assign permissions to a user:

1. Create the user.
2. Create a role or (for cluster-wide access) a cluster role.
3. Create a role binding (or cluster role binding) to assign the role to the user.

Managing users

To change the password for a user:

```
sensuctl user change-password USERNAME
```

To disable a user:

```
sensuctl user disable USERNAME
```

To re-enable a disabled user:

```
sensuctl user reinstate USERNAME
```

User specification

Attributes

username

description	The name of the user. Cannot contain special characters.
required	true
type	String
example	`"username": "alice"`

password

description	The user's password. Passwords must have at least eight characters.
required	true
type	String
example	`"password": "P@ssw0rd!"`

groups

description	Groups to which the user belongs.
required	false
type	Array
example	`"groups": ["dev", "ops"]`

disabled

description	The state of the user's account.
required	false
type	Boolean
default	false
example	`"disabled": false`

User example

The following example is in `wrapped-json` format for use with `sensuctl create`.

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "username": "alice",
    "password": "P@ssw0rd!",
    "disabled": false,
    "groups": ["ops", "dev"]
  }
}
```

Groups

A group is a set of users within Sensu.Groups can be assigned one or more roles and inherit all permissions from each role assigned to them.Users can be assigned to one or more groups.Groups are not a resource type within Sensu; you can create and manage groups only within user definitions.

Default group

Sensu includes a default `cluster-admins` group that contains the default `admin` user and a `system:agents` group used internally by Sensu agents.

Assigning 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]]
```

Removing 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 controlling access to Sensu resources. **Roles** specify permissions for resources within a namespace while **cluster roles** can include permissions for cluster-wide resources.You can use roles bindings to assign roles to user and groups.To avoid re-creating commonly used roles in each namespace, create a cluster role and use a role binding (not a cluster role binding) to restrict permissions within a specific namespace.

To create and manage roles cluster-wide, configure sensuctl as the default `admin` user or create a cluster role with `roles` permissions.To create and manage roles within a namespace, create a role with `roles` permissions within that namespace.

Cluster roles

Cluster roles can specify access permissions for cluster-wide resources like users and namespaces as well as namespaced resources like checks and handlers. They can also be used to grant access to namespaced resources across all namespaces (needed to run `sensuctl check list --all-namespaces`, for example) when used in conjunction with cluster role bindings.Cluster roles use the same specification as roles and can be managed using the same sensuctl commands with `cluster-role` substituted for `role`.

To create and manage cluster roles, configure sensuctl as the default `admin` user or create a cluster role with permissions for `clusterroles`.

Default roles

Every Sensu backend includes:

Role name	Type	Description
`cluster-admin`	`ClusterRole`	Full access to all resource types across namespaces, including access to cluster-wide resource types.
`admin`	`ClusterRole`	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`	`ClusterRole`	Read and write access to most resources 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).
`view`	`ClusterRole`	Read-only permission to most resource types with the exception of roles and role bindings. You can apply this cluster role within a namespace by using a role binding (not a cluster role binding).
`system:agent`	`ClusterRole`	Used internally by Sensu agents. You can configure an agent's user credentials using the `user` and `password` agent configuration flags.

Viewing roles

You can use sensuctl to see a list of roles within Sensu:

```
sensuctl role list
```

To see the permissions and scope for a specific role:

```
sensuctl role info admin
```

To view cluster roles, use the `cluster-role` command:

```
sensuctl cluster-role list
```

Creating a role

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
```

Once you've create the 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 the following role binding.

```
sensuctl role-binding create prod-admin-oncall --role=prod-admin --group=oncall
```

Creating a cluster-wide role

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
```

Managing roles

You can use sensuctl to view, create, edit, and delete roles.To use any of these commands with cluster roles, substitute the `cluster-role` command for the `role` command.

To edit a role:

```
sensuctl edit roles [ROLE-NAME] [flags]
```

To delete a role:

```
sensuctl role delete [ROLE-NAME]
```

To get help managing roles with sensuctl:

```
sensuctl role help
```

Role and cluster role specification

Role attributes

name	
description	Name of the role
required	true
type	String
example	`"name": "admin"`

namespace	
description	Namespace the role is restricted to. This attribute is not available for cluster roles.
required	false
type	String
example	`"namespace": "production"`

rules	
description	The rulesets that a role applies.

required	true
type	Array

example	<pre>"rules": [
	  {
	  "verbs": ["get", "list"],
	  "resources": ["checks"],
	  "resource_names": [""]
	  }
	]</pre>

Rule attributes

A rule is an explicit statement which grants a particular permission to a resource.

verbs

description	The permissions to be applied by the rule: `get`, `list`, `create`, `update`, or `delete`.
required	true
type	Array
example	<pre>"verbs": ["get", "list"]</pre>

resources

description	The type of resource that the rule has permission to access. Roles can only access namespaced resource types while cluster roles can access namespaced and cluster-wide resource types. See resource types for

	available types.
required	true
type	Array
example	`"resources": ["checks"]`

resource_names

description	Specific resource names that the rule has permission to access. Resource name permissions are only available for `get`, `delete`, and `update` verbs.
required	false
type	Array
example	`"resource_names": ["check-cpu"]`

Role example

The following example is in `wrapped-json` format for use with `sensuctl create`.

```
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "namespaced-resources-all-verbs",
    "namespace": "default"
  },
  "spec": {
```

```
    "rules": [
      {
        "resource_names": [],
        "resources": [
          "assets", "checks", "entities", "events", "filters", "handlers",
          "hooks", "mutators", "rolebindings", "roles", "silenced"
        ],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
```

Cluster role example

The following example is in `wrapped-json` format for use with `sensuctl create`.

```
{
  "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"
        ],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
```

Role bindings and cluster role bindings

A **role binding** assigns a **role** or **cluster role** to users and groups within a namesapce.A **cluster role binding** assigns a **cluster role** to users and groups across namespaces and resource types.

To create and manage role bindings within a namespace, create a role with `rolebindings` permissions within that namespace, and log in by configuring sensuctl.

Cluster role bindings

Cluster roles bindings can assign a cluster role to users and groups.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 cluster role bindings, configure sensuctl as the default `admin` user or create a cluster role with permissions for `clusterrolebindings`.

Viewing role bindings

You can use sensuctl to see a list of role bindings within Sensu:

```
sensuctl role-binding list
```

To see the details for a specific role binding:

```
sensuctl role-binding info [BINDING-NAME]
```

To see a list of cluster role bindings:

```
sensuctl cluster-role-binding list
```

Creating a role binding

You can use sensuctl to see a create a role binding that assigns a role:

```
sensuctl role-binding create [NAME] --role=NAME [--user=username] [--
group=groupname]
```

Or a role binding that assigns a cluster role:

```
sensuctl role-binding create [NAME] --cluster-role=NAME [--user=username] [--
group=groupname]
```

To create a cluster role binding:

```
sensuctl cluster-role-binding create [NAME] --cluster-role=NAME [--
user=username] [--group=groupname]
```

Managing role bindings

You can use sensuctl to see a list, create, and delete role bindings and cluster role bindings.To use any of these commands with cluster roles, substitute the `cluster-role-binding` command for the `role-binding` command.

To delete a role binding:

```
sensuctl role-binding delete [ROLE-NAME]
```

To get help managing role bindings with sensuctl:

```
sensuctl role-binding help
```

Role binding and cluster role binding specification

roleRef	
description	References a role in the current namespace or a cluster role.
required	true
type	Hash
example	`"roleRef": { "type": "Role", "name": "event-reader" }`

subjects	
description	The users or groups being assigned.
required	true
type	Array
example	`"subjects": [ { "type": "User", "name": "alice" } ]`

`roleRef` specification

type	
description	`Role` for a role binding or `ClusterRole` for a cluster role binding.
required	true
type	String
example	`"type": "Role"`

name	
description	The name of the role or cluster role being assigned.
required	true
type	String
example	`"name": "event-reader"`

`subjects` specification

type	
description	`User` for assigning a user or `Group` for assigning a group.
required	true
type	String

example	`"type": "User"`

name	
description	Username or group name.
required	true
type	String
example	`"name": "alice"`

Role binding example

The following example is in `wrapped-json` format for use with `sensuctl create`.

```
{
  "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"
```

```
        }
      ]
    }
}
```

Cluster role binding example

The following example is in `wrapped-json` format for use with `sensuctl create`.

```
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "cluster-admin"
  },
  "spec": {
    "role_ref": {
      "name": "cluster-admin",
      "type": "ClusterRole"
    },
    "subjects": [
      {
        "name": "cluster-admins",
        "type": "Group"
      }
    ]
  }
}
```

Role and role binding examples

The following role and role binding give a `dev` group access to create and manage Sensu workflows within the `default` namespace.

```
{
```

```
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "workflow-creator",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
        "resource_names": [],
        "resources": ["checks", "hooks", "filters", "events", "filters",
"mutators", "handlers"],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "dev-binding",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "workflow-creator",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "dev",
        "type": "Group"
      }
    ]
  }
}
```

Example workflows

- Assigning user permissions within a namespace
- Assigning group permissions within a namespace
- Assigning group permissions across all namespaces

Assigning user permissions within a namespace

To assign permissions to a user:

1. Create the user.
2. Create a role.
3. Create a role binding to assign the role to the user.

For example, the following configuration creates a user `alice`, a role `default-admin`, and a role binding `alice-default-admin`, giving `alice` full permissions for namespaced resource types within the `default` namespace.You can add these resources to Sensu using `sensuctl create`.

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice"
  }
}
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
```

```
        "resource_names": [],
        "resources": [
          "assets", "checks", "entities", "events", "filters", "handlers",
          "hooks", "mutators", "rolebindings", "roles", "silenced"
        ],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "alice-default-admin",
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "alice",
        "type": "User"
      }
    ]
  }
}
```

Assigning group permissions within a namespace

To assign permissions to group of users:

1. Create at least once user assigned to a group .
2. Create a role.
3. Create a role binding to assign the role to the group.

For example, the following configuration creates a user `alice` assigned to the group `ops`, a role `default-admin`, and a role binding `ops-default-admin`, giving the `ops` group full permissions for namespaced resource types within the `default` namespace.You can add these resources to Sensu using `sensuctl create`.

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
    "disabled": false,
    "username": "alice"
  }
}
{
  "type": "Role",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin",
    "namespace": "default"
  },
  "spec": {
    "rules": [
      {
        "resource_names": [],
        "resources": [
          "assets", "checks", "entities", "events", "filters", "handlers",
          "hooks", "mutators", "rolebindings", "roles", "silenced"
        ],
        "verbs": ["get", "list", "create", "update", "delete"]
      }
    ]
  }
}
{
  "type": "RoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-default-admin",
```

```
    "namespace": "default"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "Role"
    },
    "subjects": [
      {
        "name": "ops",
        "type": "Group"
      }
    ]
  }
}
```

PRO TIP: To avoid re-creating commonly used roles in each namespace, create a cluster role and use a role binding to restrict permissions within a specific namespace.

Assigning group permissions across all namespaces

To assign cluster-wide permissions to group of users:

1. Create at least once 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`.

```
{
  "type": "User",
  "api_version": "core/v2",
  "metadata": {},
  "spec": {
```

```
    "disabled": false,
    "username": "alice",
    "groups": ["ops"]
  }
}
{
  "type": "ClusterRole",
  "api_version": "core/v2",
  "metadata": {
    "name": "default-admin"
  },
  "spec": {
    "rules": [
      {
        "resource_names": [],
        "resources": [
          "assets", "checks", "entities", "events", "filters", "handlers",
          "hooks", "mutators", "rolebindings", "roles", "silenced",
          "cluster", "clusterrolebindings", "clusterroles",
          "namespaces", "users"
        ],
        "verbs": ["get", "list", "create", "update", "delete"]

      }
    ]
  }
}
{
  "type": "ClusterRoleBinding",
  "api_version": "core/v2",
  "metadata": {
    "name": "ops-default-admin"
  },
  "spec": {
    "role_ref": {
      "name": "default-admin",
      "type": "ClusterRole"
    },
    "subjects": [
      {
```

```
          "name": "ops",
          "type": "Group"
        }
      ]
    }
  }
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Sensu query expressions

- Specification
- Examples

How do Sensu query expressions work?

Sensu query expressions (**SQE**) are based on JavaScript expressions, andprovide additional functionalities for Sensu usage (like nested parameters andcustom functions) so Sensu resources can be directly evaluated. SQE shouldalways return **true** or **false**.

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

Sensu query expressions specification

Sensu query expressions are valid ECMAScript 5 (JavaScript) expressions that return **true** or **false**. Other values are not allowed. If other values arereturned, an error is logged and the filter evaluates to false.

Custom functions

- `hour`: returns the hour, in UTC and in the 24-hour time notation, of a UNIXEpoch time.

```
// event.timestamp equals to 1520275913, which is Monday, March 5, 2018 6:51:53
PM UTC
// The following expression returns true
hour(event.timestamp) >= 17
```

- `weekday`: returns a number representing the day of the week, where Sundayequals `0`, of a UNIX Epoch time.

```
// event.timestamp equals to 1520275913, which is Monday, March 5, 2018 6:51:53
PM UTC
// The following expression returns false
weekday(event.timestamp) == 0
```

Sensu query expressions examples

Evaluating an event attribute

The following example returns true if the event's entity contains a customattribute named `namespace` that is equal to `production`.

```
event.entity.namespace == 'production'
```

Evaluating an array

To evaluate an attribute that contains an array of elements, use the `.indexOf` method.The following example returns true if an entity includes the subscription `system`.

```
entity.subscriptions.indexOf('system') >= 0
```

Evaluating the day of the week

The following example returns true if the event occurred on a weekday.

```
weekday(event.timestamp) >= 1 && weekday(event.timestamp) <= 5
```

Evaluating office hours

The following example returns true if the event occurred between 9 AM and 5 PMUTC.

```
hour(event.timestamp) >= 9 && hour(event.timestamp) <= 17
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Silencing

How does silencing work?

Silencing entries are created on an ad-hoc basis via `sensuctl`. When silencingentries are successfully created, they are assigned a `name` in the format `$SUBSCRIPTION:$CHECK`, where `$SUBSCRIPTION` is the name of a Sensu entitysubscription and `$CHECK` is the name of a Sensu check. Silencing entries can beused to silence checks on specific entities by taking advantage of per-entitysubscriptions, for example: `entity:$ENTITY_NAME`. When the check name and/orsubscription described in a silencing entry match an event and a handler use the `not_silenced` built-in filter, this handler will not be executed.

These silencing entries are persisted in the Sensu data store. When the Sensuserver processes subsequent check results, matching silencing entries areretrieved from the store. If one or more matching entries exist, the event isupdated with a list of silenced entry names. The presence of silencing entriesindicates that the event is silenced.

When creating a silencing entry, a combination of check and subscription can bespecified, but only one or the other is strictly required.

For example, when a silencing entry is created specifying only a check, its namewill contain an asterisk (or wildcard) in the `$SUBSCRIPTION` position. Thisindicates that any event with a matching check name will be marked as silenced,regardless of the originating entities' subscriptions.

Conversely, a silencing entry which specifies only a subscription will have aname with an asterisk in the `$CHECK` position. This indicates that any eventwhere the originating entities' subscriptions match the subscription specifiedin the entry will be marked as silenced, regardless of the check name.

Silencing specification

Silenced entry names

Silencing entries must contain either a subscription or check name, and areidentified by the combination of `$SUBSCRIPTION:$CHECK`. If a check orsubscription is not provided, it will be substituted with a wildcard (asterisk): `$SUBSCRIPTION:*` or `*:$CHECK`.

Top-level attributes

type	
description	Top-level attribute specifying the `sensuctl create` resource type. Silencing entries should always be of type `Silenced`.
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"type": "Silenced"`

api_version	
description	Top-level attribute specifying the Sensu API group and version. For silencing entries in Sensu backend version 5.0, this attribute should

	always be `core/v2`.
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	String
example	`"api_version": "core/v2"`

metadata

description	Top-level collection of metadata about the silencing entry, including the `name` and `namespace` as well as custom `labels` and `annotations`. The `metadata` map is always at the top level of the silencing entry definition. This means that in `wrapped-json` and `yaml` formats, the `metadata` scope occurs outside the `spec` scope. See the metadata attributes reference for details.
required	Required for silencing entry definitions in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs
example	`"metadata": { "name": "appserver:mysql_status", "namespace": "default", "labels": { "region": "us-west-1" } }`

spec

description	Top-level map that includes the silencing entry spec attributes.

required	Required for silencing entries in `wrapped-json` or `yaml` format for use with `sensuctl create`.
type	Map of key-value pairs

example	<pre>"spec": {
	  "expire": -1,
	  "expire_on_resolve": false,
	  "creator": "admin",
	  "reason": null,
	  "check": null,
	  "subscription": "entity:i-424242",
	  "begin": 1542671205
	}</pre>

Spec attributes

check

description	The name of the check the entry should match
required	true, unless `subscription` is provided
type	String
example	<pre>"check": "haproxy_status"</pre>

subscription

description	The name of the subscription the entry should match
required	true, unless `check` is provided

type	String
example	`"subscription": "entity:i-424242"`

begin

description	Time at which silence entry goes into effect, in epoch.
required	false
type	Integer
example	`"begin": 1512512023`

expire

description	Number of seconds until this entry should be deleted.
required	false
type	Integer
default	-1
example	`"expire": 3600`

expire_on_resolve

description	If the entry should be deleted when a check begins return OK status (resolves).

required	false
type	Boolean
default	false
example	`"expire_on_resolve": true`

creator

description	Person/application/entity responsible for creating the entry.
required	false
type	String
default	null
example	`"creator": "Application Deploy Tool 5.0"`

reason

description	Explanation for the creation of this entry.
required	false
type	String
default	null
example	`"reason": "rebooting the world"`

Metadata attributes

name	
description	Silencing identifier generated from the combination of a subscription name and check name.
required	false - This value cannot be modified.
type	String
example	`"name": "appserver:mysql_status"`

namespace	
description	The Sensu RBAC namespace that this silencing entry belongs to.
required	false
type	String
default	`default`
example	`"namespace": "production"`

labels	
description	Custom attributes to include with event data, which can be queried like regular attributes.

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

example	<pre>"labels": {
	  "environment": "development",
	  "region": "us-west-2"
	}</pre>

annotations	
description	Arbitrary, non-identifying metadata to include with event data. You can use annotations to add data that helps people or external tools interacting with Sensu.
required	false
type	Map of key-value pairs. Keys and values can be any valid UTF-8 string.
default	`null`

example	<pre>"annotations": {
	  "managed-by": "ops",
	  "slack-channel": "#monitoring",
	  "playbook": "www.example.url"
	}</pre>

Examples

Silence all checks on a specific entity

Assume a Sensu entity `i-424242` which we wish to silence any alerts on. We'lldo this by taking advantage of per-entity subscriptions:

```
{
  "type": "Silenced",
  "api_version": "core/v2",
  "metadata": {
    "name": "entity:i-424242:*",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "expire": -1,
    "expire_on_resolve": false,
    "creator": "admin",
    "reason": null,
    "check": null,
    "subscription": "entity:i-424242",
    "begin": 1542671205
  }
}
```

Silence a specific check on a specific entity

Following on the previous example, silence a check named `check_ntp` on entity `i-424242`, ensuring the entry is deleted once the underlying issue has beenresolved:

```
{
  "subscription": "entity:i-424242",
  "check": "check_ntp",
  "expire_on_resolve": true
}
```

The optional `expire_on_resolve` attribute used here indicates that when theserver processes a matching check from the specified entity with status OK, thissilencing entry will automatically be removed.

When used in combination with other attributes (like `creator` and `reason`), thisprovides Sensu operators with a method of acknowledging that they have receivedan alert, suppressing additional notifications, and automatically clearing thesilencing entry when the check status returns to normal.

Silence all checks on entities with a specific subscription

In this case, we'll completely silence any entities subscribed to `appserver`.Just as in the example of silencing all checks on a specific entity, we'llcreate a silencing entry specifying only the `appserver` subscription:

```
{
  "subscription": "appserver"
}
```

Silence a specific check on entities with a specific subscription

Assume a check `mysql_status` which we wish to silence, running on Sensuentities with the subscription `appserver`:

```
{
  "subscription": "appserver",
  "check": "mysql_status"
}
```

Silence a specific check on every entity

To silence the check `mysql_status` on every entity in our infrastructure,regardless of subscriptions, we only need to provide the check name:

```
{
  "check": "mysql_status"
}
```

Deleting silencing entries

To delete a silencing entry, you will need to provide its name. Subscription onlysilencing entry names will be similar to this:

```
{
  "name": "appserver:*"
}
```

Check only silencing entry names will be similar to this:

```
{
  "name": "*:mysql_status"
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Tessen

Tessen is available in version 5.5.0 and later.See the upgrade guide to upgrade your Sensu installation, and visit the latest reference documentation.

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation

Products

- Sensu Core
- Sensu Enterprise

Solutions

- For Containers
- For Cloud
- For Nagios

Pricing

Tokens

Tokens are placeholders included 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 re-using the check definition.

How do tokens work?

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 will return an error, which is logged and sent to the Sensu backend message transport. Checks with token matching errors will not be executed.

Managing entity labels

You can use token substitution with any defined entity attributes, including custom labels.See the entity reference for information on managing entity labels for proxy entities and agent entities.

Sensu token specification

Sensu Go uses the Go template package to implement token substitution.Sensu Go token substitution uses double curly braces around the token, and a dot before the attribute to be substituted, such as: `{{ .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. Nested Sensu entity attributes can be accessed via dot notation (ex: `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`

Token substitution default values

In the event that an attribute is not provided by the entity, a token's defaultvalue will be substituted. Token default values are separated by a pipe character and the word `default` (`| default`), and can be used to provide a "fallback value" for entities that are missing a specified token attribute.

- `{{.labels.url | default "https://sensu.io"}}` would be replaced with a custom label called `url`. If no such attribute called `url` is included in the entity definition, the default (or fallback) value of `https://sensu.io` will be used to substitute the token.

Unmatched tokens

If a token is unmatched during check preparation, the agent check handler will return an error, and the check will not be executed. Unmatched token errors will look similar to the following:

```
error: unmatched token: template: :1:22: executing "" at <.system.hostname>: map has no entry for key "System"
```

Check config token errors will be logged by the agent, and sent to Sensu backend message transport as a check failure.

Examples

Token substitution for check thresholds

In this example check configuration, the `check-disk-usage.go` command accepts `-w` (warning) and `-c` (critical)arguments to indicate the thresholds (as percentages) for creating warning or critical events. If no token substitutions are provided by an entity configuration, Sensu will use default values to create a warning event at 80% disk capacity (i.e. `{{ .labels.disk_warning | default 80 }}`), and a critical event at 90% capacity (i.e. `{{ .labels.disk_critical | default 90 }}`).

```
{
  "type": "CheckConfig",
  "api_version": "core/v1",
  "metadata": {
    "name": "check-disk-usage",
    "namespace": "default",
    "labels": null,
    "annotations": null
  },
  "spec": {
    "command": "check-disk-usage.rb -w {{.labels.disk_warning | default 80}} -c
{{.labels.disk_critical | default 90}}",
    "handlers": [],
    "high_flap_threshold": 0,
    "interval": 10,
    "low_flap_threshold": 0,
    "publish": true,
    "runtime_assets": null,
    "subscriptions": [
    "staging"
    ],
    "proxy_entity_name": "",
    "check_hooks": null,
    "stdin": false,
    "ttl": 0,
    "timeout": 0,
    "env_vars": null
  }
}
```

The following example entity would provide the necessaryattributes to override the

`.labels.disk_warning` and `labels.disk_critical` tokens declared above.


```
{
  "type": "Entity",
  "api_version": "core/v2",
  "metadata": {
    "name": "example-hostname",
    "namespace": "default",
    "labels": {
      "disk_warning": "80",
      "disk_critical": "90"
    },
    "annotations": null
  },
  "spec": {
    "entity_class": "agent",
    "system": {
      "hostname": "example-hostname",
      "os": "linux",
      "platform": "centos",
      "platform_family": "rhel",
      "platform_version": "7.4.1708",
      "network": {
        "interfaces": [
          {
            "name": "lo",
            "addresses": [
              "127.0.0.1/8",
              "::1/128"
            ]
          },
          {
            "name": "enp0s3",
            "mac": "08:00:27:11:ad:d2",
            "addresses": [
              "10.0.2.15/24",
              "fe80::26a5:54ec:cf0d:9704/64"
            ]
          },
          {
```

```
            "name": "enp0s8",
            "mac": "08:00:27:bc:be:60",
            "addresses": [
              "172.28.128.3/24",
              "fe80::a00:27ff:febc:be60/64"
            ]
          }
        ]
      },
      "arch": "amd64"
    },
    "subscriptions": [
      "entity:example-hostname",
      "staging"
    ],
    "last_seen": 1542667231,
    "deregister": false,
    "deregistration": {},
    "user": "agent",
    "redact": [
      "password",
      "passwd",
      "pass",
      "api_key",
      "api_token",
      "access_key",
      "secret_key",
      "private_key",
      "secret"
    ]
  }
}
```

About Sensu

The Sensu monitoring event pipeline empowers businesses to automate their monitoring workflows and gain deep visibility into their multi-cloud infrastructure, from Kubernetes to bare metal. Companies like Sony, Box.com, and Activision rely on Sensu to help deliver value faster, at scale.

Navigation