

SENSU MONITORING SETUP

Introduction

Effective monitoring of resources and services, and the ability to react to them in a timely fashion is a linchpin of DevOps engineering.

Monitoring takes many forms and there is no such thing as a finished monitoring system. Just when you think you have everything monitored, you will find an mysterious edge case that will cause issues if not correctly accounted for. On the other side, you need to ensure that your monitoring is accurate. Nothing kills a monitoring platform quicker than having to look past a wall of false alerts. People start to ignore alerts, sooner rather than later, and you end up with an outage, which you could have seen coming but did not.

What is Sensu and Why is it different?

Sensu is a project of “Heavy Water” which have gone further and have started to re-evaluate the core principles to rearrange the monitoring infrastructure. (<https://sensuapp.org>)

- Publication/Subscription

Sensu takes a different approach to many monitoring solutions and, rather than using a master/client solution, it uses a Publication/Subscription model using a message queue. This approach makes it far simpler to configure monitoring for a large number of services and is useful for environments that contain a large amount of ephemeral hosts.

- Ease of adding new clients

By allowing clients to subscribe simply and easily to a set of checks, it makes it much easier to roll out new clients; they simply start the client with the correct subscription, and start running and sending results back to the master to process.

- Divide we stand

Other than utilizing a message queue, Sensu allows for custom checks to be written relatively easily and with flexibility. This allows the responsibilities of writing checks to be split evenly

among team members, both application and infrastructure developers. It should help ensure that your monitoring coverage is as broad as possible.

Enough Talkings

We will show you how we set up, configured, and rolled out checks using SENSU. Configuring SENSU to alert us via the two most important avenues: SMS and e-mail was very crucial. A summary of topics covered is as follows:-

- Installing a SENSU server
- Installing a SENSU client
- Installing check prerequisites
- Finding community checks
- Adding a DNS check
- Adding a disk check
- Adding a RAM check
- Adding a process check
- Adding a CPU check
- Creating e-mail alerts
- Creating SMS alerts
- Using Ansible to install SENSU

Installing a SENSU server

Setting up SENSU and its prerequisites was pretty straightforward, and it took us little time to set it up and configure some initial checks. This Recipe will guide you to setup a Debian based SENSU-Master as we call it is will show you how to install SENSU Core and Uchiwa as its dashboard.

Getting ready

Launch an Ubuntu (14.04 or higher) server to host both SENSU Core and Uchiwa.

How to do it

The following steps will demonstrate how to install a SENSU server, RabbitMQ, and the Uchiwa panel:

- Our first step is to Install RabbitMQ, which requires Erlang to be installed. Use the following command to install Erlang:

```
$sudo apt-get update && sudo apt-get -y install erlang-nox
```

- Execute the following command to add the APT repository to your `/etc/apt/sources.list.d`:

```
$echo 'deb http://www.rabbitmq.com/debian/ testing main' |sudo tee  
/etc/apt/sources.list.d/rabbitmq.list
```

- **Note:- To avoid warnings about unsigned packages, add this public key to your trusted key list using apt-key**

```
wget -O- https://www.rabbitmq.com/rabbitmq-release-signing-key.asc  
|  
sudo apt-key add -
```

- Run the following command to update the package list:

```
sudo apt-get update
```

- Install `rabbitmq-server` package:

```
sudo apt-get install rabbitmq-server
```

Configure RabbitMQ access controls

- **Note:-Access to RabbitMQ is restricted by access controls (e.g. username and password). For SENSU services to connect to RabbitMQ a RabbitMQ virtual host (vhost) and user credentials are required.**

- **Create a dedicated RabbitMQ vhost for Sensu**

```
sudo rabbitmqctl add_vhost /sensu
```

- **Create a RabbitMQ user for Sensu**

```
sudo rabbitmqctl add_user sensu secret
```

```
sudo rabbitmqctl set_permissions -p /sensu sensu ".*" ".*" ".*"
```

Configure system limits on Linux

- **Note:** By default, most Linux operating systems will limit the maximum number of file handles a single process is allowed to have open to 1024. RabbitMQ recommends adjusting this number to 65536

To adjust this limit, please edit the configuration file found at `/etc/default/rabbitmq-server` by uncommenting the last line in the file:-

```
ulimit -n 65536
```

To verify that the RabbitMQ open file limits, please run:-

```
rabbitmqctl status
```

Setting up RabbitMQ is complete and should work now; Redis is another prerequisite required . We can install Redis using the following command:

```
$sudo apt-get install redis-server
```

Now that we have installed and configured RabbitMQ and Redis, we can install Sensu Core. First, we add the Sensu package signing key using the following command:

```
$wget -q https://sensu.global.ssl.fastly.net/apt/pubkey.gpg -O- | sudo apt-key  
add -
```

Get the codename of the Ubuntu/Debian release on your system and then We can then add the repository with this command:

```
. /etc/os-release && echo $VERSION
```

```
export CODENAME=your_release_codename_here
```

```
echo "deb https://sensu.global.ssl.fastly.net/apt $CODENAME main" |sudo tee  
/etc/apt/sources.list.d/sensu.list
```

Finally, we install the Sensu package. This installs the server, API, and client as a bundle:

```
$ sudo apt-get update && sudo apt-get install sensu
```

Now we have installed the Sensu server, we can configure it. First, copy the client certificate we created earlier in place with the following command:

```
$ sudo mkdir -p /etc/sensu/ssl && sudo cp client/cert.pem client/key.pem  
/etc/sensu/ssl
```

Next, we configure our Sensu server's basic connectivity.

Create a new file called `config.json` under the directory of `/etc/sensu` and insert the following content:

```
{  
  "rabbitmq": {  
    "ssl": {  
      "cert_chain_file": "/etc/sensu/ssl/cert.pem",  
      "private_key_file": "/etc/sensu/ssl/key.pem"
```

```
{,
  "host": "localhost",
  "port": 5671,
  "vhost": "/sensu",
  "user": "sensu",
  "password": "<<password>>"
},
"redis": {
  "host": "localhost"
},
"api": {
  "port": 4567
}
}
```

- Note: Ensure that you replace `<<password>>` with the password you chose while setting up the Sensu RabbitMQ user.

Start the Sensu services using the following commands:

```
$ sudo service sensu-server start
$ sudo service sensu-api start
```

TIP::: Notice that we're not starting the Sensu client in this instance; this is to keep this recipe focused. I thoroughly encourage you to configure a client on your Sensu master and monitor it as with any other host.

Next, we are going to install `uchiwa`. Uchiwa is an elegant and easy to use dashboard for Sensu, designed to be used for information radiators, such as TVs. Install Uchiwa using the following command:

```
$ sudo apt-get install uchiwa
```

Now that we have installed Uchiwa, we need to configure it; edit the file `/etc/sensu/uchiwa.json` and ensure that it has the following content:

```
{
  "sensu": [
    {
      "name": "<<site description>>",
      "host": "<<sensuserver>>",
      "port": 4567,
      "timeout": 5
    }
  ],
  "uchiwa": {
    "host": "0.0.0.0",
    "port": 3000,
    "interval": 5
  }
}
```

- Note: Ensure that you replace `<<site description>>` and `<<sensuserver>>` with the correct values. The site description can be set to a description of your choice. Ensure that the [sensu](#) server is set to the DNS name or IP address of your Sensu server.

Start the service using the following command:

```
$ sudo service uchiwa start
```

Within your browser, navigate to your Sensu server on port **3000** and you should be able to see a screen similar to the following screenshot:



See also

- You can find further installation details for Sensu at <https://sensuapp.org/docs/0.20/installation-overview>
- You can find installation details of Uchiwa at <http://docs.uchiwa.io/en/latest/getting-started/>

Installing a Sensu client

Once we installed the Sensu server, we needed to install the Sensu client to run checks and report the data back to the server. The Sensu client subscribes to the RabbitMQ virtual host and listens for checks to be published in a subscription to which the client belongs. When a check is published, the client runs the assigned check and publishes the results back onto the RabbitMQ; from here, the Sensu server then processes the check results.

Getting ready

Launch an Ubuntu (14.04 or higher) instance to act as the Sensu client.

How to do it...

- The Sensu Core package used to install the Sensu client includes the client, server, and API package. First we add the Sensu package signing key using the following command:

```
$ wget -q http://repos.sensuapp.org/apt/pubkey.gpg -O- | sudo apt-key add
```


-

Then we add the repository:

```
$ echo "deb http://repos.sensuapp.org/apt sensu main" | sudo tee  
/etc/apt/sources.list.d/sensu.list
```

- Finally, we can now install the Sensu package using the following command:

```
$ sudo apt-get update && sudo apt-get install sensu
```

TIP::: If you have already read the *How to install a Sensu server* recipe, then you might have noticed that the steps are the same for both client and server. This is because the Sensu package is an omnibus package that contains everything; you only start the services you want.

- Next, we need to copy the client key into place. You can start by creating a directory to place the certificates with this command:

```
$ sudo mkdir -p /etc/sensu/ssl
```

Copy the following files from the keys you created when you created your Sensu server and into the directory you created in the step above:

```
client/cert.pem  
client/key.pem
```

- Now, we can configure the Sensu client. First, create a file in `/etc/sensu` called `config.json` and insert the following content:

```
{  
  "rabbitmq": {  
    "ssl": {  
      "cert_chain_file": "/etc/sensu/ssl/cert.pem",
```

```

        "private_key_file": "/etc/sensu/ssl/key.pem"
    },
    "host": "<sensumaster>",
    "port": 5671,
    "vhost": "/sensu",
    "user": "sensu",
    "password": "<password>"
},
"redis": {
    "host": "localhost"
},
"api": {
    "port": 4567
}
}

```

- Note that you need to replace the values of `<<sensumaster>>` and `<<password>>` with the IP or name of your Sensu Master and Sensu password, respectively.

- Now that we have configured the general Sensu connectivity, we can configure the client specific settings. Create a new file under `/etc/sensu/conf.d` called `client.json` and insert the following content:

```

{
  "client": {
    "name": "sensuhost",
    "address": "<ip address>",
    "subscriptions": [ "common" ]
  }
}

```

- It's worth going over this slim piece of configuration. The first part of the configuration defines the name that is displayed for this host (instance to be monitored) when reporting the check results; I suggest that this should be the DNS name of the client for easy identification. The `address` field allows you to define an IP address that the client reports as its originating address. The `subscriptions` field allows you to add subscriptions for checks. I recommend that you have a

common set of checks that all hosts should respond to; these can be things, such as disk space, CPU usage, RAM usage, and so on.

TIP:::Subscriptions are a key part of using Sensu, and are a fantastic organizational tool. I generally recommend using a common subscription for the usual suspects, such as RAM and CPU checks. You can use a role description for other subscriptions, such as a subscription called `nginx_server`, or `haproxy_lb`.

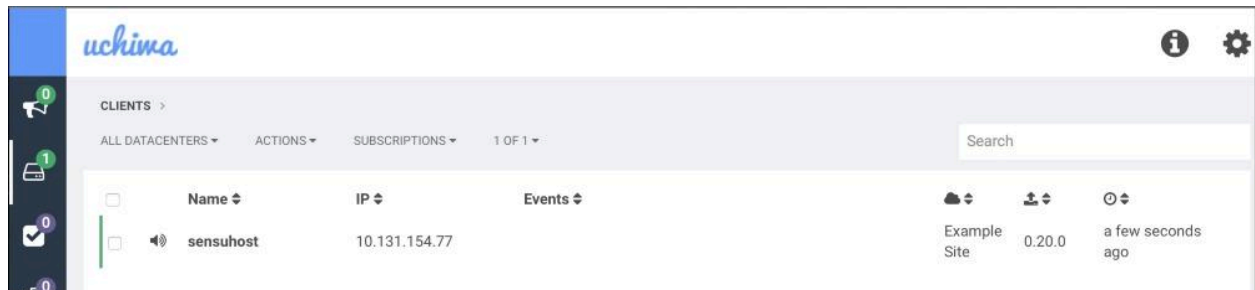
- Now that we have configured our client, we are ready to start the Sensu client service. Start it by issuing the following command:

```
$ service sensu-client start
```

On your `sensu master`, log into your `uchiwa` panel and select this icon:



This takes you to the client-listing page. Once there, you should be able to see your new host listed and it should look something similar to the following screenshot:



See also

You can find further details about the client installation at <https://sensuapp.org/docs/0.20/install-sensu-client>.

Installing check prerequisites

Sensu checks are generally written using Ruby and there is broad support for the language throughout Sensu. However, this means that there are certain dependencies on Ruby for some checks.

Getting ready

For this recipe, you will require an Ubuntu 14.04 server with Sensu installed.

TIP::: This recipe requires you to install development tools to compile the native Ruby extensions that some checks require. If having development tools on hosts contravenes your security policies, I recommend that you use a tool, such as FPM (<https://github.com/jordansissel/fpm>) to build the checks on a build machine and then re-package for distribution.

How to do it

To install the various packages, issue the following command:

```
$ sudo apt-get install -y ruby ruby-dev build-essential
```

Finding community checks

Once the Sensu client and server are installed, it's now time to add the checks to be monitored for any issue. By default, the Sensu client reports nothing; it is up to you to add any relevant checks to make it useful.

Sensu checks can be written in any language as long as it returns the correct response to the server via RabbitMQ; however, they are generally written either in Bash or more commonly in Ruby. Luckily, the Sensu community has contributed a great many open source checks to the project. These can be installed, thus saving you from having to create your own and they cover many of the common check scenarios.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host to act as the Sensu check host and a Sensu server to connect to. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

You can find the Sensu community checks at <http://sensu-plugins.io/plugins/>.

Each link should take you to a Github page containing the code for that particular check, and it should have the additional documentation on the usage of the check.

See also

- You can find further details of the Sensu community checks at <http://sensu-plugins.io/plugins/>
- You can find details of the Sensu check format at <https://sensuapp.org/docs/latest/checks>

Adding a DNS check

Almost every application has external dependencies, such as databases, Redis caches, e-mail servers, and so on. Although it is generally reliable, DNS can occasionally cause problems and, at first glance, it can be difficult to diagnose. By adding a check that constantly checks the DNS record, you can be assured that these dependencies are available.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host to act as the Sensu check-host and a Sensu server to connect to. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's start by adding DNS check to track DNS records:

- For this recipe, we're going to install the `sensu-plugins-dns`. Use the following command to install the new plugin:

```
$ sudo gem install sensu-plugins-dns
```

- You can test if the plugin has been installed successfully by issuing the following command:

```
$ check-dns.rb -d www.packtpub.com
```

- You should see a response like the following:

```
DNS OK: Resolved www.packtpub.com A records
```

- On the Sensu server, create a new file called `web_check.json` under the directory `/etc/sensu/conf.d` and insert the following content:

```
{
  "checks": {
    "check_google": {
      "command": "/usr/local/bin/check-dns.rb -d google.com",
      "interval": 60,
```

```

        "subscribers": [ "web_check" ]
    },
    "check_yahoo": {
        "command": "/usr/local/bin/check-dns.rb -d yahoo.com",
        "interval": 60,
        "subscribers": [ "web_check" ]
    },
    "check_fail": {
        "command": "/usr/local/bin/check-dns.rb -d sdfdsssf.com",
        "interval": 60,
        "subscribers": [ "web_check" ]
    }
}
}

```

- Once you have entered the configuration, restart the Sensu server by issuing the following command:

```
$ service sensu-server restart
```

- Next, we need to configure our client to subscribe to the checks. Edit the `client.json` file located within `/etc/sensu/conf.d` and to reflect the following code:

```

{
  "client": {
    "name": "sensuhost",
    "address": "<<SENSUCLIENTIP>>",
    "subscriptions": [ "common", "web_check" ]
  }
}

```

- Notice the additional subscription. When you restart the client, it will subscribe to a set of checks that publish themselves for `web_check` clients to run. Restart the Sensu client by issuing the following command:

```
$ service sensu-client restart
```

- Your checks should now be running; however, it may take a few minutes for them to show up. This is due to the checks being placed on the MQ and being checked at the interval specified (60

seconds in the above example). To check, log on to Uchiwa on your Sensu master and select the following icon on the left-hand side:



- You should see the check to which we have deliberately given a nonsense address:

CLIENTS >





ALL DATACENTERS ▾

ACTIONS ▾

SUBSCRIPTIONS ▾

1 OF 1 ▾

Search

<input type="checkbox"/>	Name ⇅	IP ⇅	Events ⇅	 ⇅	 ⇅	 ⇅
<div><div></div><div> sensuhost</div></div>	10.131.154.77	DNS CRITICAL: Could not resolve sdfdsssf.com	Example Site	0.20.0	a few seconds ago	

- You can also check in `/var/log/sensu/sensu-server.log` and locate the line that resembles the following:
 - ```
{"timestamp":"2015-07-14T16:55:14.404660-0400","level":"info","message":"processing event","event":{"id":"c12ebe42-62ed-454f-a074-49c71c3c8f7a","client":{"name":"sensuhost","address":"10.131.154.77","subscriptions":["common","web_check"],"version":"0.20.0","timestamp":1436907305},"check":{"command":"/usr/local/bin/check-dns.rb -d sdfdsssf.com","interval":60,"subscribers":["web_check"],"name":"check_fail","issued":1436907314,"executed":1436907314,"duration":0.262,"output":"DNS CRITICAL: Could not resolve"}}
```



```
sdfdsssf.com\n", "status":2, "history":["2"], "total_state_change":0}, "occurrences":1, "action":"create"}}
```

### See also

You can find further details of the Sensu DNS check at <https://github.com/sensu-plugins/sensu-plugins-dns>.

## Adding a disk check

Disk checks are a critical part of infrastructure monitoring. A full disk can cause processes to fail, servers to slow down, and logs to be lost. Providing alerts for disk issues in a timely manner is vital to the smooth running of any infrastructure.

This recipe shows you how to install the community disk check and add it to a subscription called common.

### Getting ready

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

### How to do it...

Let's install the community disk check:

1. First, we install the disk check `gem` using the Gem package manager:
  2. 

```
$ sudo gem install sensu-plugins-disk-checks
```
3. **TIP**
4. This gem installs many additional disk based checks in addition to space usage, allowing you to check for issues such as SMART alerts, and so on. You can see the details at: <https://github.com/sensu-plugins/sensu-plugins-disk-checks>.
5. Now we can configure the check configuration. On the Sensu master, create a new file called `disk_checks.json` under the `/etc/sensu/conf.d` directory and insert the following content:

```
6. {
 "checks": {
 "check_disk_usage": {
 "command": "/usr/local/bin/check-disk-usage.rb -w 75 -c 90",
 "interval": 60,
 "subscribers": ["common"]
 }
 }
}
```

7. The configuration above makes use of the disk check plugging, and uses the `-w` and `-c` switches. These switches are relatively common amongst Nagios-style checks and allows you to set a warning and a critical threshold. In this case, I'm using a warning at 75% and a critical alert at 90%. This is very useful as it allows us to use different alert types based on the threshold; for instance, a warning could trigger an e-mail and a critical alert could send an SMS. Read the plugin documentation to find details of what thresholds you can set and how to set them.

8. On the client side, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

```
9. {
 "client": {
 "name": "sensuhost",
 "address": "10.131.154.77",
 "subscriptions": ["common", "web_check"]
 }
}
```

10. To check that the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

```
11. {"timestamp":"2015-07-14T17:26:42.689883-0400","level":"info","message":"publishing check request","payload":{"name":"check_disk_usage","issued":1436909202,"command":"/usr/local/bin/check-disk-usage.rb -w 75"},"subscribers":["common"]}
```

## See also

You can find more details for the community disk checks at <https://github.com/sensu-plugins/sensu-plugins-disk-checks>.

Having sufficient RAM available for a server is a crucial part of running a performant service. When memory resources run short, the application either runs slow, if the OS is forced to use swap space, or in extremes can cause applications to crash.

This recipe demonstrates how to use Sensu to monitor that sufficient free RAM is present on a monitored system.

## Getting ready

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

## How to do it...

Let's add the RAM check:

1. We need to install the [sensu-plugins-memory-checks gem](#); this installs an executable for the RAM check:
  2. `$ sudo gem install sensu-plugins-memory-checks`
3. Now we can configure the [config](#) check. On the Sensu master, create a new file called [ram\\_checks.json](#) under the [/etc/sensu/conf.d](#) directory and insert the following content:
  4. 

```
{
 "checks": {
 "check_ram": {
 "command": "/usr/local/bin/check-ram.rb -w 70 -c 95",
 "interval": 60,
 "subscribers": ["common"]
 }
 }
}
```
5. Again, note the use of the `-w` and `-c` switches; these set the thresholds in percentage used that needs to trigger an alert.
6. On the client, edit the file called [client.json](#) within [/etc/sensu/conf.d](#) and ensure that the following code is present:
  7. 

```
{
 "client": {
 "name": "sensuhost",
```

```
 "address": "10.131.154.77",
 "subscriptions": ["common", "web_check"]
 }
}
```

8. To determine that the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:
  9. 

```
{ "timestamp": "2015-07-14T17:43:28.066609-0400", "level": "warn", "message": "config file applied changes", "file": "/etc/sensu/conf.d/check_ram.json", "changes": { "checks": { "check_ram_usage": [null, { "command": "/usr/local/bin/check-ram.rb", "interval": 60, "subscribers": ["common"] }] } } }
```

### See also

For further details on the Sensu memory checks, see the following page  
<https://github.com/sensu-plugins/sensu-plugins-memory-checks>.

## Adding a process check

One important item to monitor is if a process is actually running on a system. It's little use knowing that you have plenty of disk and CPU resources, but not realizing that your apache server has fallen over. Sensu can be used to monitor the key processes that are running on your server and it can alert you if a process has gone AWOL.

This recipe shows you how to check if the `ssh` process is running on any host subscribed to the common subscriptions; however, the same technique can be used to monitor any process.

### Getting ready...

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

### How to do it...

This recipe will show you how to install the Sensu process check plugin and how to configure it to monitor a running process.

1. First, we install the process check [gem](#) using the Gem package manager. Use the the following command to install the plugin:

2. `$ sudo gem install sensu-plugins-process-checks`

3. Now, we can configure the check configuration. On the Sensu master, create a new file called `sshd_process_check.json` under the `/etc/sensu/conf.d` directory and insert the following content:

4. 

```
{
 "checks": {
 "check_sshd_usage": {
 "command": "/usr/local/bin/check-process.rb -p 'sshd -D'",
 "interval": 60,
 "subscribers": ["common"]
 }
 }
}
```

5. This check makes use of the `-p` switch to allow us to specify a process that we wish to monitor; this should be the full string of the running process (notice that, in the preceding example, I have added the `-D` switch that the process runs with).

6. On the client, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

7. 

```
{
 "client": {
 "name": "sensuhost",
 "address": "10.131.154.77",
 "subscriptions": ["common", "web_check"]
 }
}
```

8. To determine if the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

9. 

```
{"timestamp":"2015-07-14T18:13:48.464091-0400","level":"info","message":"processing
event","event":{"id":"f1326a4f-87c2-49a7-8b28-70dfa3e9836b","client":
{"name":"sensuhost","address":"10.131.154.77","subscriptions":["comm
on","web_check"],"version":"0.20.0","timestamp":1436912025},"check":
{"command":"/usr/local/bin/check-process.rb -p 'sshd
-D',"interval":60,"subscribers":["common"],"name":"check_sshd_usage",
```

```
"issued":1436912028,"executed":1436912028,"duration":0.128,"output":
"CheckProcess OK: Found 1 matching processes; cmd /sshd
-D/\n","status":0,"history":["1","1","1","1","0"],"total_state_change":0},"
occurrences":4,"action":"resolve"}}
```

## See also

You can find further details of the process checks at  
<https://github.com/sensu-plugins/sensu-plugins-process-checks>.

## Adding a CPU check

Having sufficient CPU resources is a vital part of running a performant service and it is hard to spot without sufficient monitoring. Using Sensu to alert when CPU usage is running high, you will be able to deal with slow running processes before the customer notices.

## Getting ready

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

## How to do it...

Let's add a CPU usage check:

1. First, we install the CPU check gem using the Gem package manager. Use the following command to install the plugin:
  2. `$ sudo gem install sensu-plugins-cpu-checks`
3. Now we can configure the check configuration. On the Sensu master, create a new file called `cpu_check.json` under the `/etc/sensu/conf.d` directory and insert the following content:
  4. 

```
{
 "checks": {
 "check_cpu_usage": {
 "command": "/usr/local/bin/check-cpu.rb",
 "interval": 60,
```

```
 "subscribers": ["common"]
 }
}
}
```

5. On the client, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

```
6. {
 "client": {
 "name": "sensuhost",
 "address": "10.131.154.77",
 "subscriptions": ["common", "web_check"]
 }
}
```

7. To determine that the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

```
8. {"timestamp":"2015-07-15T16:24:26.800371-0400","level":"info","message":"publishing check request","payload":{"name":"check_cpu","issued":1436991866,"command":"/usr/local/bin/check-cp.rb"},"subscribers":["common"]}
```

## See also

You can find further details of the process checks at <https://github.com/sensu-plugins/sensu-plugins-cpu-checks>.

## Creating e-mail alerts

Although you can view your Sensu alerts using the Uchiwa panel, it's unlikely that you will have your eyes glued to the TV at all times. Instead, you need to give Sensu the ability to alert you in a more interactive fashion, and one of the most tried and trusted methods is via e-mail. In today's world of laptops, Smartphones and tablets, it's a rare time indeed when you are not able to receive e-mails. This recipe will show you how to configure the Sensu e-mail plugin to allow you to receive e-mails whenever an alert is triggered.

## Getting ready

For this recipe, you will need a configured Sensu server and Sensu client. You should also have at least one check configured. You will also need an SMTP server that can relay mail. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

### How to do it...

Let's create an e-mail alert:

1. First, you can use Ruby's gem to install the mail plugin using the following command:
  2. `$ gem install sensu-plugins-mailer`
3. Now, we can configure the mail plugin. Create a new file called `plugin_mailer.json` within `/etc/sensu/conf.d` and insert the following content:
  4. 

```
{
 "mailer": {
 "admin_gui": "http://<sensuserver>/#/events",
 "mail_from": "<<fromaddress>>",
 "mail_to": "<<toaddress>>",
 "smtp_address": "<<smtpaddress>>",
 "smtp_username": "<<smtpusername>>",
 "smtp_password": "<<smtppassword>>",
 "smtp_port": "587",
 "smtp_domain": "<<smtpdomain>>"
 }
}
```
5. Ensure that you replace the values inside the angle brackets with the relevant information for your e-mail setup. The admin `gui` is simply a link to the `uchiwa` panel, so fill in the address of your Sensu server.
6. Now the mailer plugin is configured, we can create the handler.
7. You can combine the plugin and handler settings into the same file, but it's better practice to keep them separate.
8. A handler is an executable piece of code that is triggered by an event sent via a Plugin; you can think of plugins as raising alerts and handlers as dealing with how to distribute the event to end users. Sensu allows you to configure many different handlers, which allows you to be flexible in how you are alerted. You may wish to e-mail some checks, others you might want to send via SMS, and still others you might want to allow for a combination of the two; handler definitions



allow you to define these. To create the [handler](#) definition for the mailer, create a new file called [mail.json](#) under the [/etc/sensu/handlers](#) and insert the following content:

```
9. {
 "handlers": {
 "mailer": {
 "type": "pipe",
 "command": "/usr/local/bin/handler-mailer.rb"
 }
 }
}
```

10. This has created a new handler called mailer that we can make available for our checks. The type of [pipe](#) is the most commonly used type of handler and outputs the contents of the Sensu event into the command. Effectively, the event is raised by a plugin, placed on the MQ, processed by the Sensu Server, and then parsed via the handler.

11. To add the handler to a check, open up a check definition and amend it to include the following code:

```
12. {
 "checks": {
 "check_cpu": {
 "command": "/usr/local/bin/check-cpu.rb",
 "interval": 60,
 "subscribers": ["common"],
 "handlers": ["mailer"]
 }
 }
}
```

13. Now, whenever an alert is triggered, you should receive an e-mail that resembles something like this:

```
14. DNS CRITICAL: Could not resolve sdfdsssf.com
Admin GUI: http://sensumaster.stunthmaster.com/#/events
Host: sensuhost
Timestamp: 2015-07-15 19:07:14 -0400
Address: 10.131.154.77
Check Name: check_fail
Command: /usr/local/bin/check-dns.rb -d sdfdsssf.com
Status: CRITICAL
Occurrences: 1
And when the check is resolved, you should see a resolution E-mail that
looks something like this:
```

Resolving on request of the API  
Admin GUI: <http://sensumaster.stunthmaster.com/#/events>  
Host: sensuhost  
Timestamp: 2015-07-19 19:15:12 -0400  
Address: 10.131.154.77  
Check Name: check\_fail  
Command: /usr/local/bin/check-dns.rb -d sdfdsssf.com  
Status: OK  
Occurrences: 2976

15. By editing the [handler-mailer.rb](#) code, you can modify this e-mail to more suit your formatting needs.

### See also

## Creating SMS alerts

Sometimes you need alerts that are more immediate than an e-mail. When a critical service goes down, you don't want to miss it because you eschewed carrying a smartphone and your laptop wasn't near by. SMS messaging is a fantastic way to send default alerts and is in many ways the spiritual successor to the pager. SMS has the advantage of being almost universal and it is virtually impossible in this day and age to find a cell phone that does not support it.

Unlike e-mail, you cannot run a local SMS server to send messages directly; instead, you need to sign up with an SMS gateway, which will route your messages to the various mobile phone providers. In this recipe, we're going to use Twilio (<https://www.twilio.com>). Twilio supports both Voice and SMS gateways and has an easy to use API. Like all SMS gateways, Twilio charges per message; however, trial accounts are available to test your integration.

### Getting ready

For this recipe, you will need a Sensu server, Sensu client, and at least one configured check. You will also need a Mobile phone to receive your test message. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

### How to do it...

Let's create SMS alerts:

1. First, signup for a new Twilio account by visiting <https://www.twilio.com/try-twilio>. It will ask you for some basic details, and will send you an e-mail to confirm the account. Ensure that you have confirmed your details, and that you can log in.
2. Once you have a Twilio account, you can install the Twilio Sensu plugin using the following command:
  3. `$ sudo gem install sensu-plugins-twilio`
4. Next, we will configure the plugin. As with the mailer plugin, this takes two forms: the handler configuration and the plugin configuration. Let's deal with the plugin configuration first: create a new file `/etc/sensu/conf.d/plugin_twilio.json` and insert the following content:
  5. 

```
{
 "twiliosms": {
 "token": "<<TWILIO_TOKEN>>",
 "sid": "<<TWILIO_SID>>",
 "number": "<<TWILIO_NUMBER>>",
 "recipients": {
 "+<<RECIPIENT_NUMBER>>": {
 "sensu_roles": ["all"],
 "sensu_checks": [],
 "sensu_level": 1
 }
 }
 }
}
```
6. There are a few things to note with this code. First, you need to have your own Twilio API and SID at hand; if you need to find them, you can find them on this page:
7. <https://www.twilio.com/user/account/settings>
8. They should be available about halfway down the page and will resemble this:

### API Credentials

Live

AccountSID

Used to exercise the REST API

ACd4704838e15e99f35203c5cde1ce82a1

AuthToken (Request a Secondary Token)

Keep this somewhere safe and secure

.....

[Learn more about REST API Credentials](#)

Test

Test AccountSID

Used to exercise the REST API

AC89427835d6d2666a39b893b7fd980f38

Test AuthToken

Keep this somewhere safe and secure

.....

[Learn more about Test Credentials](#)

9.

10. Next we need to set up the recipient number. This is an array and can contain as many recipients as you need; however, the recipient will need to be acknowledged within the Twilio panel.

11. **TIP**

12. A Twilio test account has many limitations, including a limit on the recipients.

13. Each recipient can have a different set of roles and checks that will trigger an SMS; in our example, we're leaving it as all roles to ensure that every alert will send an SMS. However, you can use this configuration item to restrict SMS alerts only to critical roles.

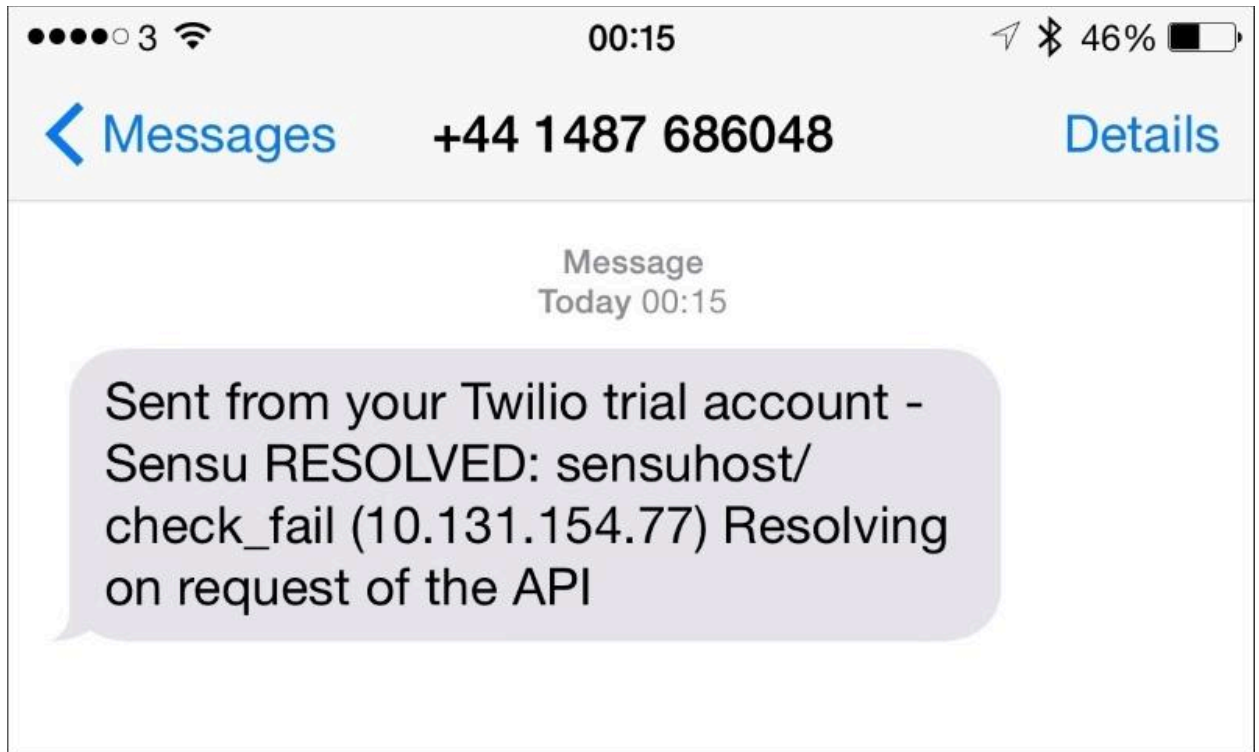
14. Next, we can configure the handler configuration. Create a new file called `/etc/sensu/handlers/plugin_twilio_sms.json` and add the following configuration:

```
15. {
 "handlers": {
 "twiliosms": {
 "type": "pipe",
 "command":
 "/var/lib/gems/1.9.1/gems/sensu-plugins-twilio-0.0.3/bin/handler-twilio-
ms.rb"
 }
 }
 }
```

16. Once you have done this, save the file and restart the Sensu server with the following command:

17. `$ sudo service sensu-server restart`

The next time you have an alert, you should receive an SMS message that looks similar to the following screenshot:



18.

#### See also

- You can find more information about Twilio at <https://www.twilio.com>
- You can find further information about the Twilio plugin at <https://github.com/sensu/sensu-community-plugins/blob/master/handlers/notification/twiliosms.rb>

#### Using Ansible to install Sensu

When rolling out on any kind of scale, it's almost certain that you will want to use automation to perform the install, especially for the clients; this allows you to roll out the changes quickly, easily, and with minimum fuss.

As with other recipes in this book, we are going to use Ansible as our automation tool of choice and rather than write a new playbook from scratch, we're going to make use of a truly excellent role available on the Ansible galaxy (<https://galaxy.ansible.com/detail#/role/279>).

If you need a refresher on Ansible, see Chapter 5, *Automation with Ansible*.

#### Getting ready

For this recipe, you will need a node to run the Ansible playbook and at least two servers: one to act as the Sensu server and the other, the Sensu client. The Sensu Server node should have RabbitMQ and Redis already installed on it. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

### TIP

Although slightly out of scope for this recipe, you can use two other Ansible roles to automate both RabbitMQ and Redis, <https://github.com/Mayeu/ansible-playbook-rabbitmq> and <https://github.com/DavidWittman/ansible-redis>, respectively.

### How to do it...

1. On the host that will act as your Ansible node, run the following command to install the Sensu role:
  2. `ansible-galaxy install Mayeu.sensu`
3. **TIP**
4. If you do not have Ansible installed in the default location, you can also clone the role and place it in your own structure; the code is available here:  
<https://github.com/Mayeu/ansible-playbook-sensu/blob/master/vagrant/site.yml>.
5. Next, let's create an inventory. I'm assuming you are using the default location for the inventory; otherwise, use the `-i` switch on the `ansible-playbook` command to specify one in the location of your choice. In the inventory, ensure that you have the following:
  6. `[sensu_servers]`  
`<<SENSUSERVERS>`  
  
`[sensu_clients]`  
`<<SENSUCLIENTS>>`
7. Where `<<SENSUSERVER>>` is the DNS name of your Sensu server and `<<SENSU_CLIENTS>>` are the DNS names of your Sensu clients.
8. Now, we need to create a new playbook; create a new file called `<<playbook>>/sensu.yml` and insert the following content:
  9. `- hosts: sensu_servers`  
`user: <<sudo_user>>`  
`vars:`

```

- sensu_install_server: true
- sensu_install_client: false
vars_files:
- group_vars/sensu.yml
roles:
- Mayeu.sensu

- hosts: sensu_clients
 user: <<sudo_user>>
 vars:
 - sensu_install_server: false
 - sensu_install_client: true
 - sensu_client_hostname: '{{ ansible_hostname }}'
 - sensu_client_address: '{{ ansible_eth0["ipv4"]["address"] }}'
 - sensu_client_subscription_names [common]
 }}'
 vars_files:
 - group_vars/sensu.yml
 roles:
 - Mayeu.sensu

```

10. Replace `<<playbook>>` with the name of the directory you wish your playbook to reside in. Also replace the `<<sudo_user>>` with a user that has sudo permissions on the servers you are connecting to.
11. As you can see, in this playbook we are defining two different plays: one for the Sensu Server and another for the Sensu clients. Note how each play references a variable file allowing us to define certain shared configuration items. We are also setting certain variables at the client level where we need differences.
12. Next, we need to create some directories to contain the files that the Ansible role will require. Use the following command to create them:
  13. `mkdir -p <playbook>/files/sensu/extensions && mkdir -p <playbook>/files/sensu/handlers && mkdir -p <playbook>/files/sensu/plugins`
14. These folders are to hold your handlers, plugins, and if you use them, your extensions.
15. **TIP**
16. You can find details of Sensu extensions at <https://sensuapp.org/docs/0.20/extensions>.

17. For instance, if you wish to add a new plugin, you will first add the plugin `ruby` file to the `<playbook>/files/sensu/plugins` directory; this will then be placed in the appropriate place on the Sensu server.
18. Next, create a folder to hold your Sensu certificates:
  19. `mkdir -p <playbook>/files/sensu/certs`
20. Now, copy your certificates into the newly created folder.
21. **TIP**
22. You can find the details for how to create the certificates in the recipe entitled *How to install a sensu server*.
23. Now, let's create the variables file for the common Sensu items. Create a new file under `<<playbook>>/group_vars` called `sensu.yml` and insert the following content:
  24. `sensu_server_rabbitmq_hostname: '<<SENSUSERVERDNSNAME>>'`  
`sensu_server_rabbitmq_user: <<sensuMQuser>>`  
`sensu_server_rabbitmq_password: <<sensuMQpassword>>`  
`sensu_server_rabbitmq_vhost: "sensu"`  
`sensu_server_api_user: <<SENSU_USER>>`  
`sensu_server_api_password: <<SENSU_PASSWORD>>`
25. The values in this file define the settings for your Sensu server and cover aspects such as the MQ to connect to the vhost, username, password, and so on. You may notice that this seems to be a short list; this is because this particular role has very sensible defaults. You can find the list of defaults on the readme at <https://galaxy.ansible.com/detail#/role/279>.
26. Now we have configured our server and client, the next step is to define some checks to run on the client. First, we configure the server to send out a request for the check to the `commonsubscription`. Insert the following into the `<<playbook>>/group_vars/sensu.yml` file:
  27. `sensu_checks:`  
`cpu_check:`  
`handler: default`  
`command: "/usr/local/bin/check-cpu.rb"`  
`interval: 60`  
`subscribers:`  
`- common`
28. You will also need to install the check-package onto the clients; you can do this by inserting the following code within the role that sets up the client. I normally recommend making this part of any common role that is used to setup all hosts:
  29. `name: "Install Sensu CPU Check Plugin"`  
`gem: name='sensu-plugins-cpu-checks' state=present`



30. Finally, we should define a handler on the Sensu server; insert the following code into the `<<playbook>>/group_vars/sensu.yml` file:

31. `sensu_handlers:`

`basic_mailer:`

`type: pipe`

`command: "mailx -s 'Sensu Alert' opsuser@opsaddress.com"`

32. The preceding handler will send a simple e-mail if there is an alert, and you can use the same technique to set up as many handlers as you like.

33. Run Ansible using the following command:

34. `ansible-playbook -K sensu.yml`

35. This should install Sensu and Uchiwa, and configure the clients with a cpu check, plus add a simple handler.

### See also

You can find details of the Sensu Ansible role at <https://github.com/Mayeu/ansible-playbook-sensu>.