Red Hat OpenStack Platform 8
Red Hat OpenStack Platform Operational Tools

Centralized Logging and Monitoring of an OpenStack Environment

OpenStack Team

Red Hat OpenStack Platform 8 Red Hat OpenStack Platform Operational Tools

Centralized Logging and Monitoring of an OpenStack Environment

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This document describes the installation and configuration of the operational tools that provide centralized logging, availability monitoring, and performance monitoring.

Table of Contents

PREFACE

Red Hat OpenStack Platform comes with an optional suite of tools designed to help operators maintain an OpenStack environment. The tools perform the following functions:

- Centralized logging
- Availability monitoring
- Performance monitoring

This document describes the preparation and installation of these tools.

> **Warning**
>
> The Red Hat OpenStack Platform Operational Tool Suite is currently a Technology Preview. For more information on Red Hat Technology Previews, see Technology Preview Features Support Scope.

CHAPTER 1. ARCHITECTURE

1.1. CENTRALIZED LOGGING

The centralized logging toolchain consists of a number of components, including:

- A Log Collection Agent (Fluentd)
- A Log Relay/Transformer (Fluentd)
- A Data Store (Elasticsearch)
- An API/Presentation Layer (Kibana)

These components and their interactions are laid out in the following diagrams:

Figure 1.1. Centralized logging architecture at a high level

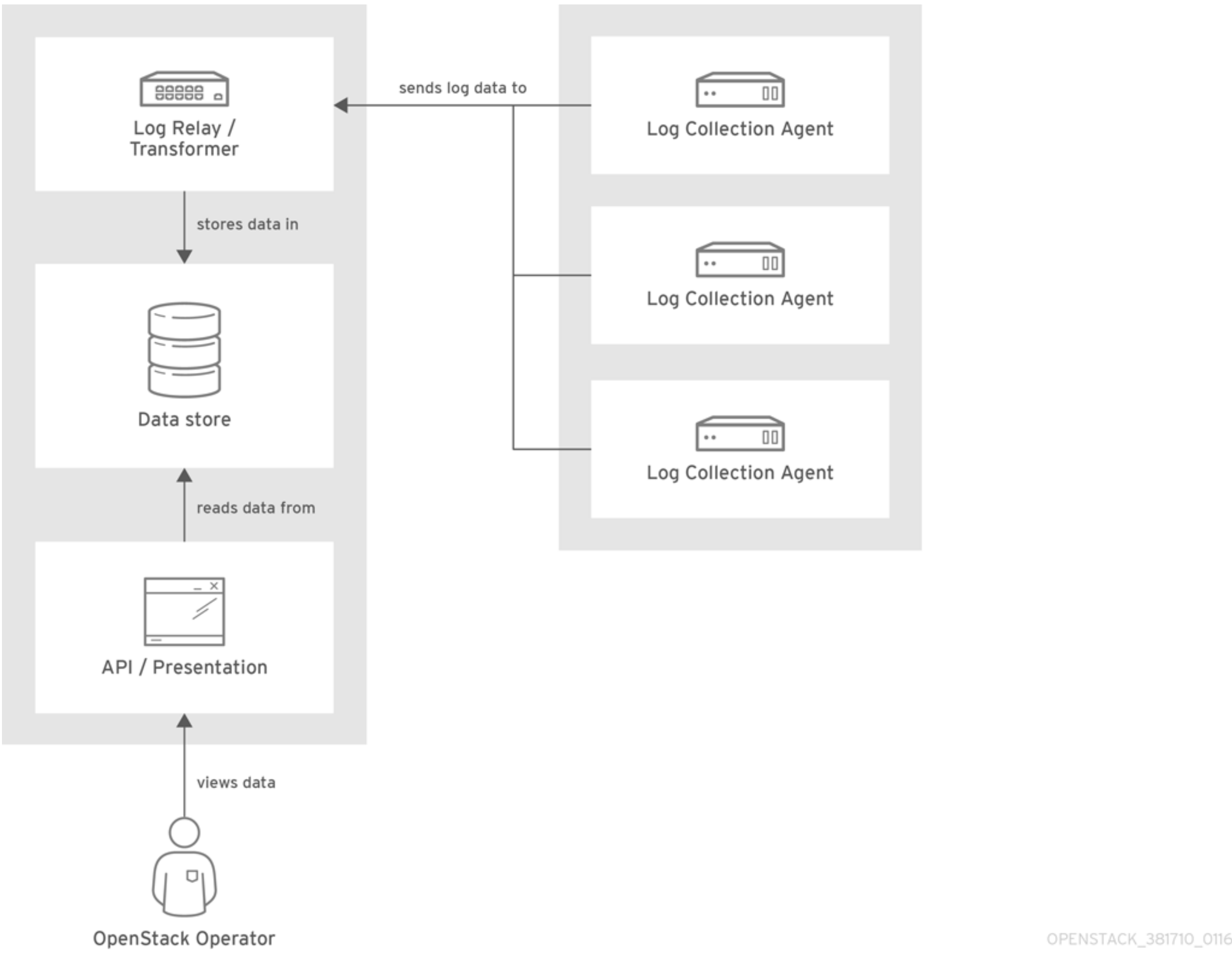


Figure 1.2. Single-node deployment for Red Hat OpenStack Platform

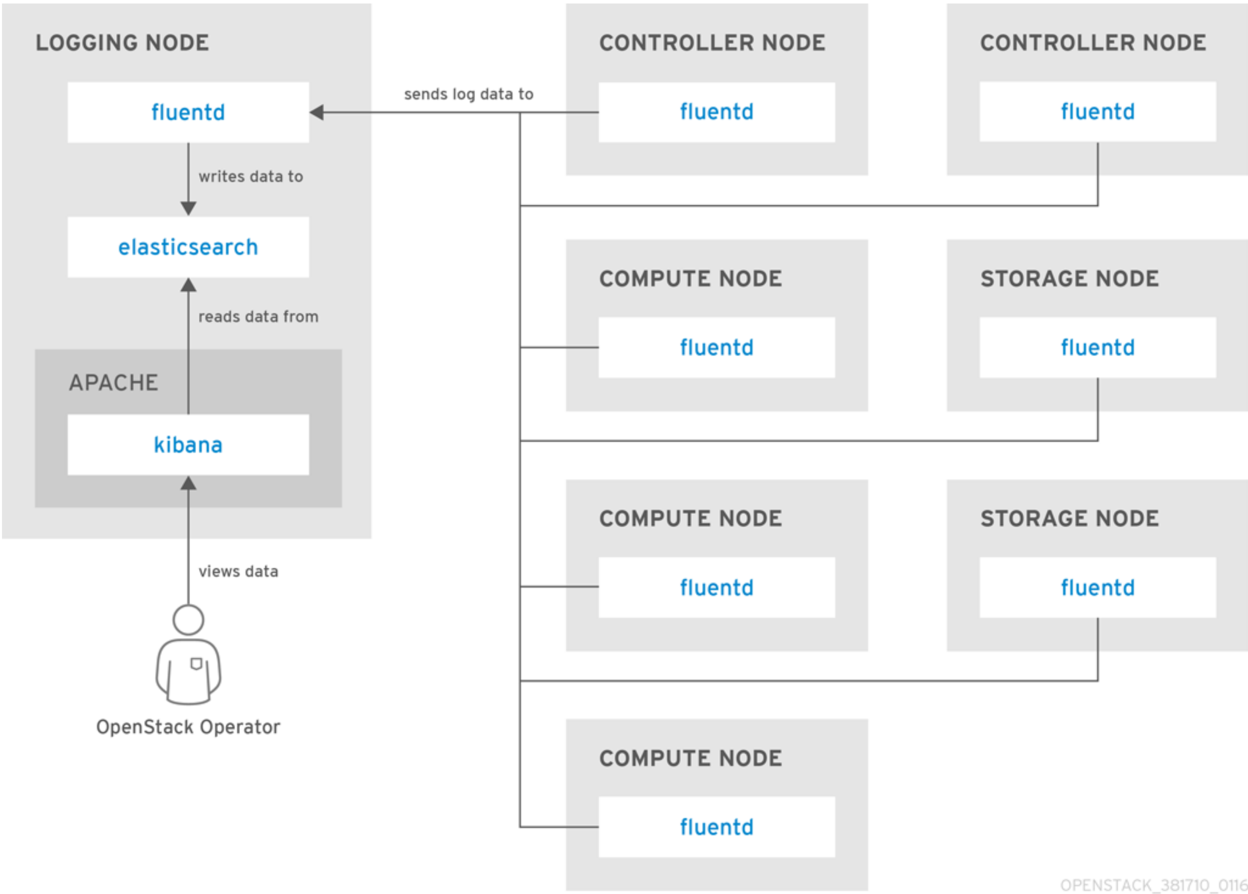
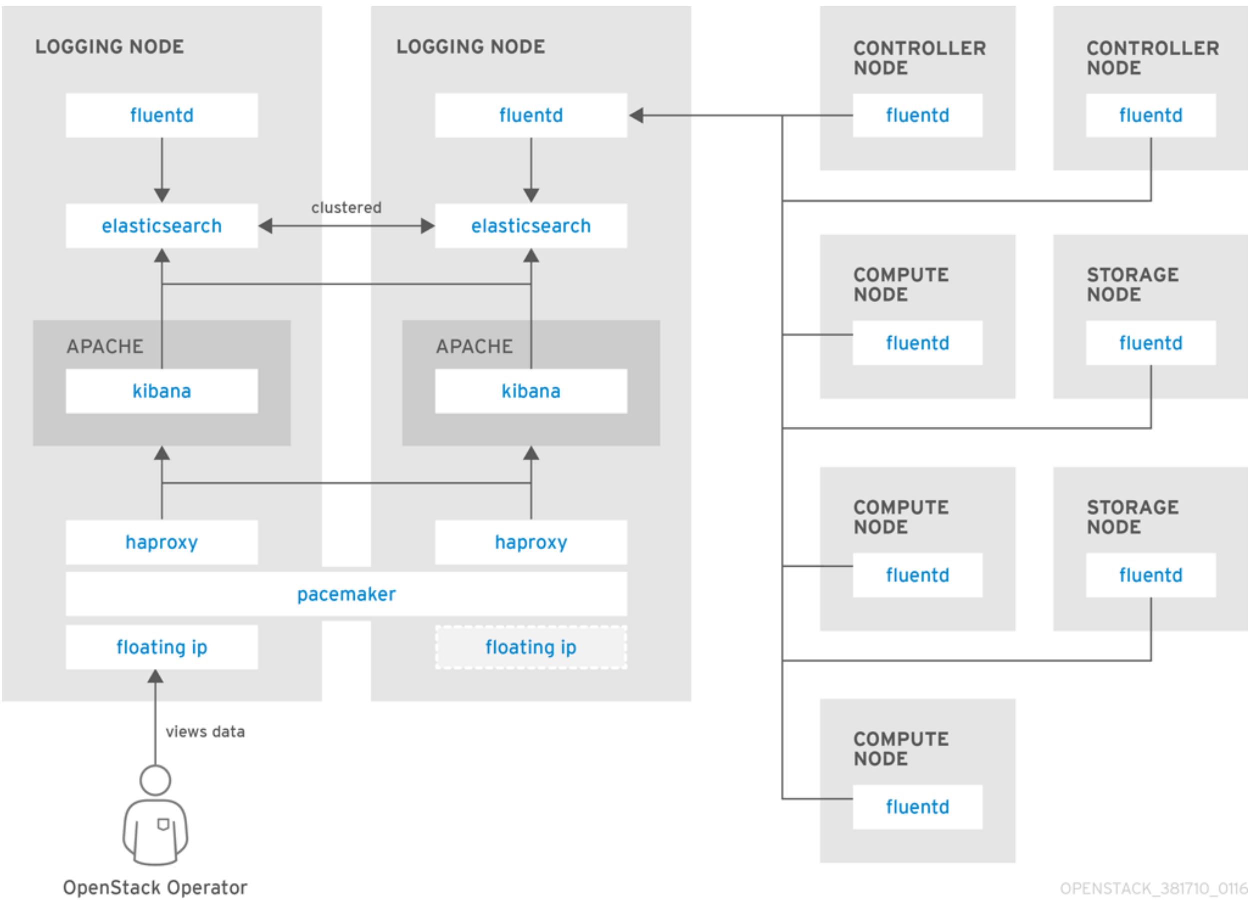


Figure 1.3. HA deployment for Red Hat OpenStack Platform



1.2. AVAILABILITY MONITORING

1.2. AVAILABILITY MONITORING

The availability monitoring toolchain consists of a number of components, including:

- A Monitoring Agent (Sensu)
- A Monitoring Relay/Proxy (RabbitMQ)
- A Monitoring Controller/Server (Sensu)
- An API/Presentation Layer (Uchiwa)

These components and their interactions are laid out in the following diagrams:

Figure 1.4. Availability monitoring architecture at a high level

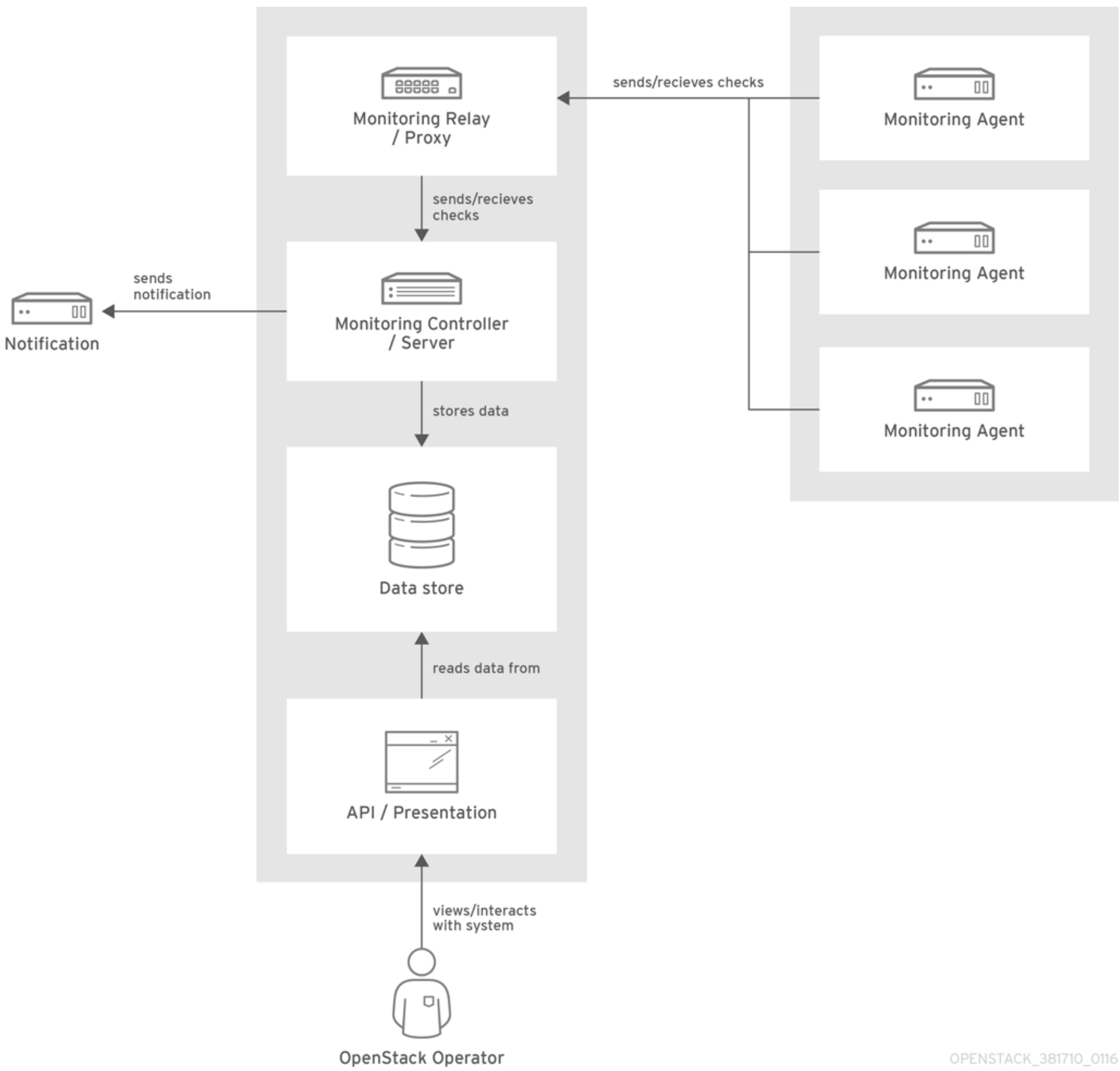


Figure 1.5. Single-node deployment for Red Hat OpenStack Platform

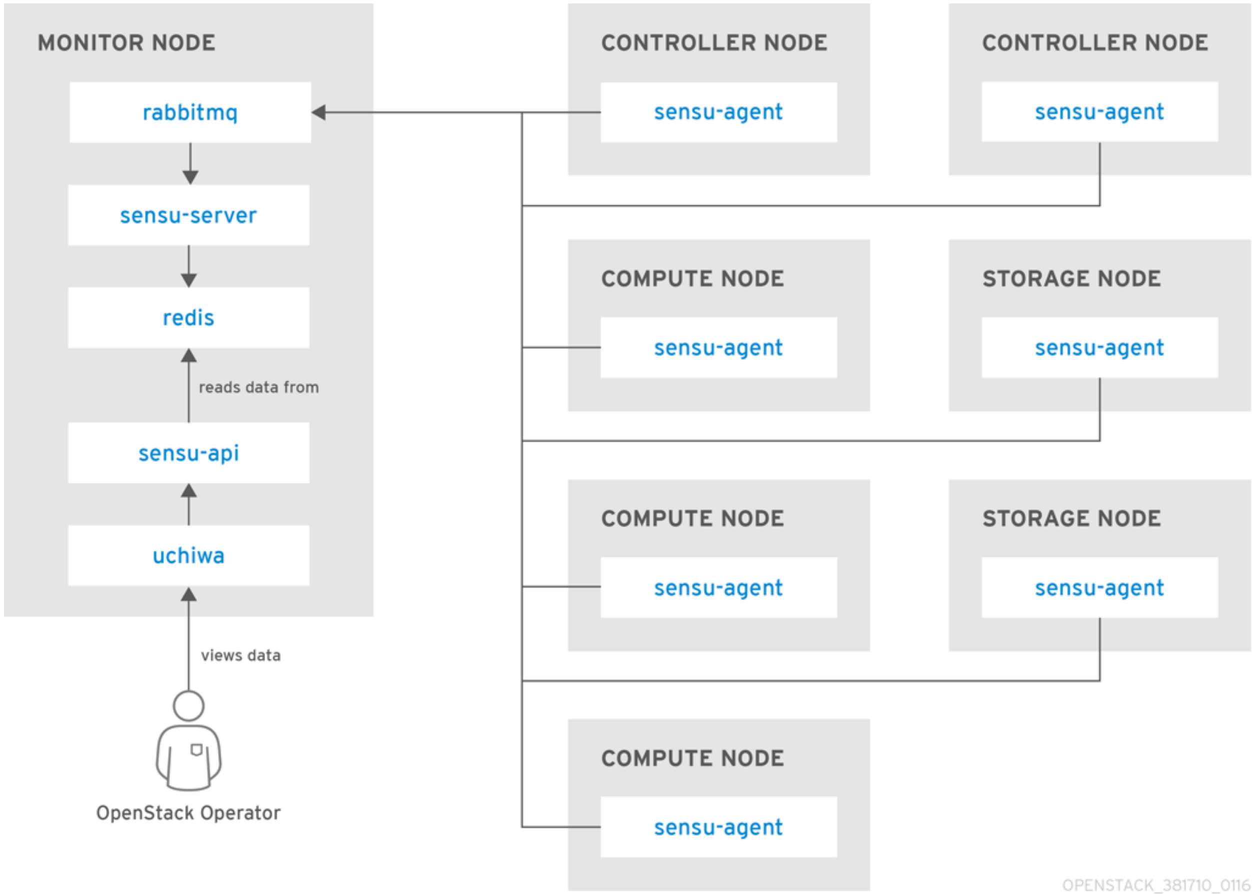
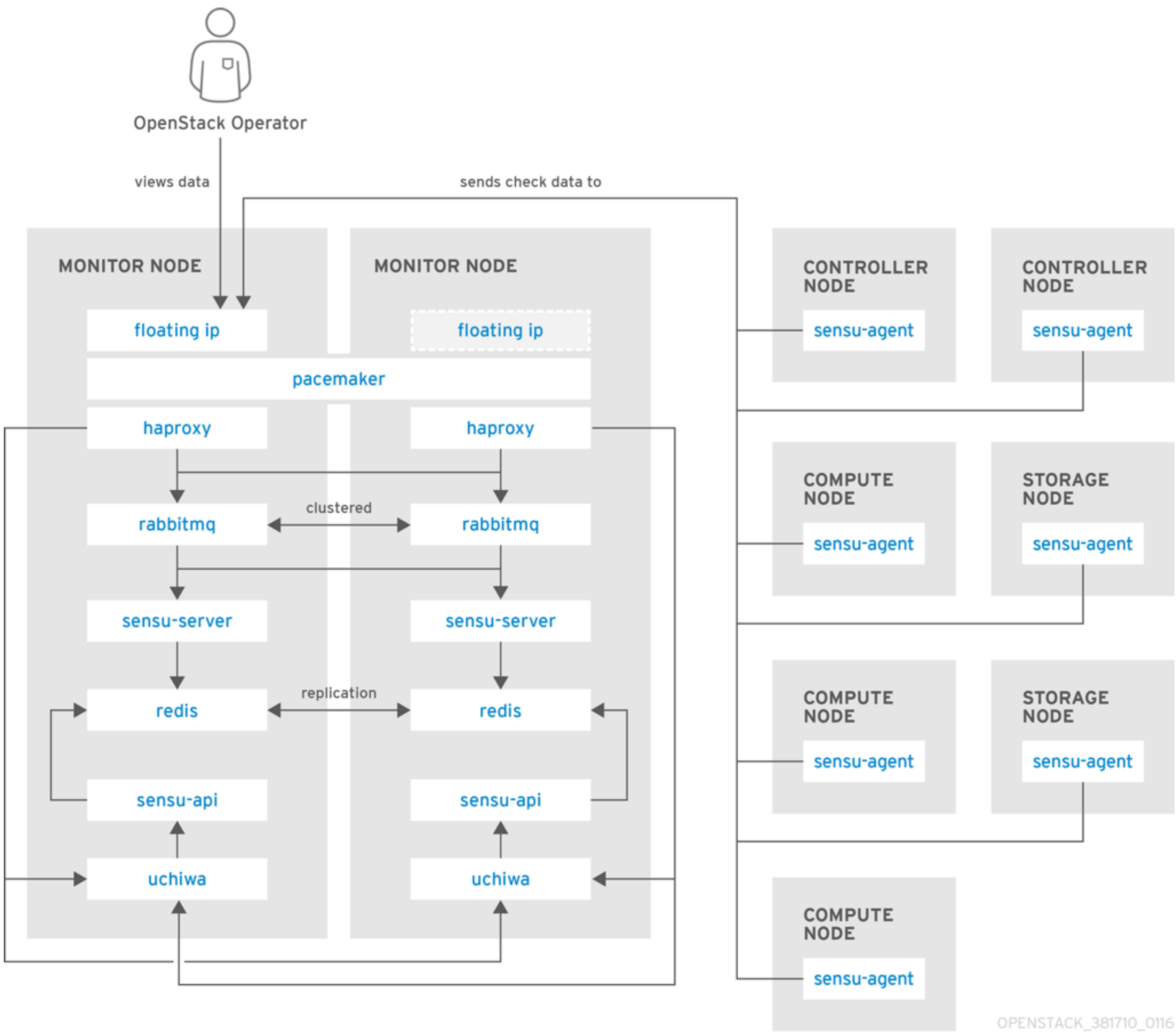


Figure 1.6. HA deployment for Red Hat OpenStack Platform



1.3. PERFORMANCE MONITORING

The performance monitoring toolchain consists of a number of components, including:

- A Collection Agent (collectd)
- A Collection Aggregator/Relay (Graphite)
- A Data Store (whisperdb)
- An API/Presentation Layer (Grafana)

These components and their interactions are laid out in the following diagrams:

Figure 1.7. Performance monitoring architecture at a high level

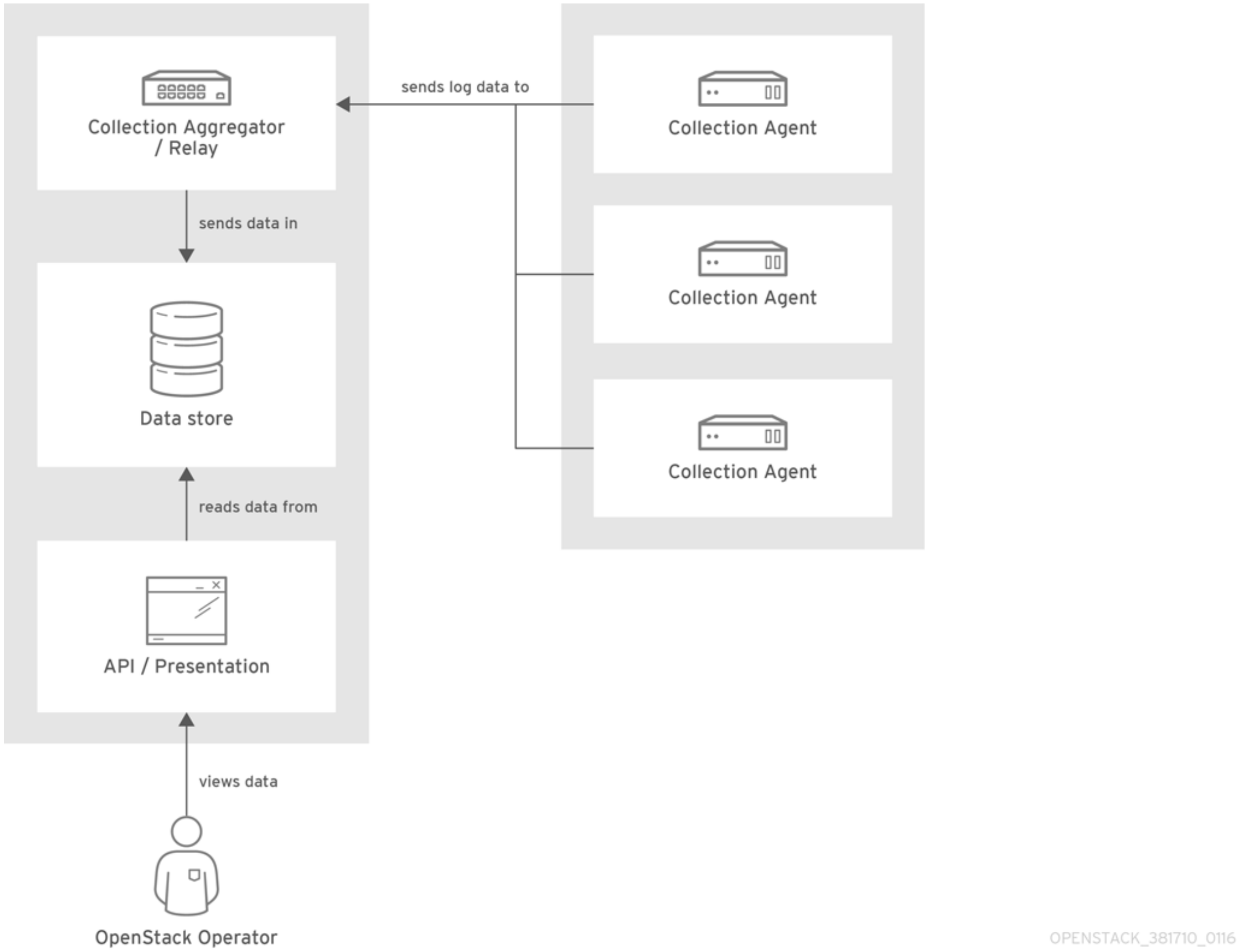


Figure 1.8. Single-node deployment for Red Hat OpenStack Platform

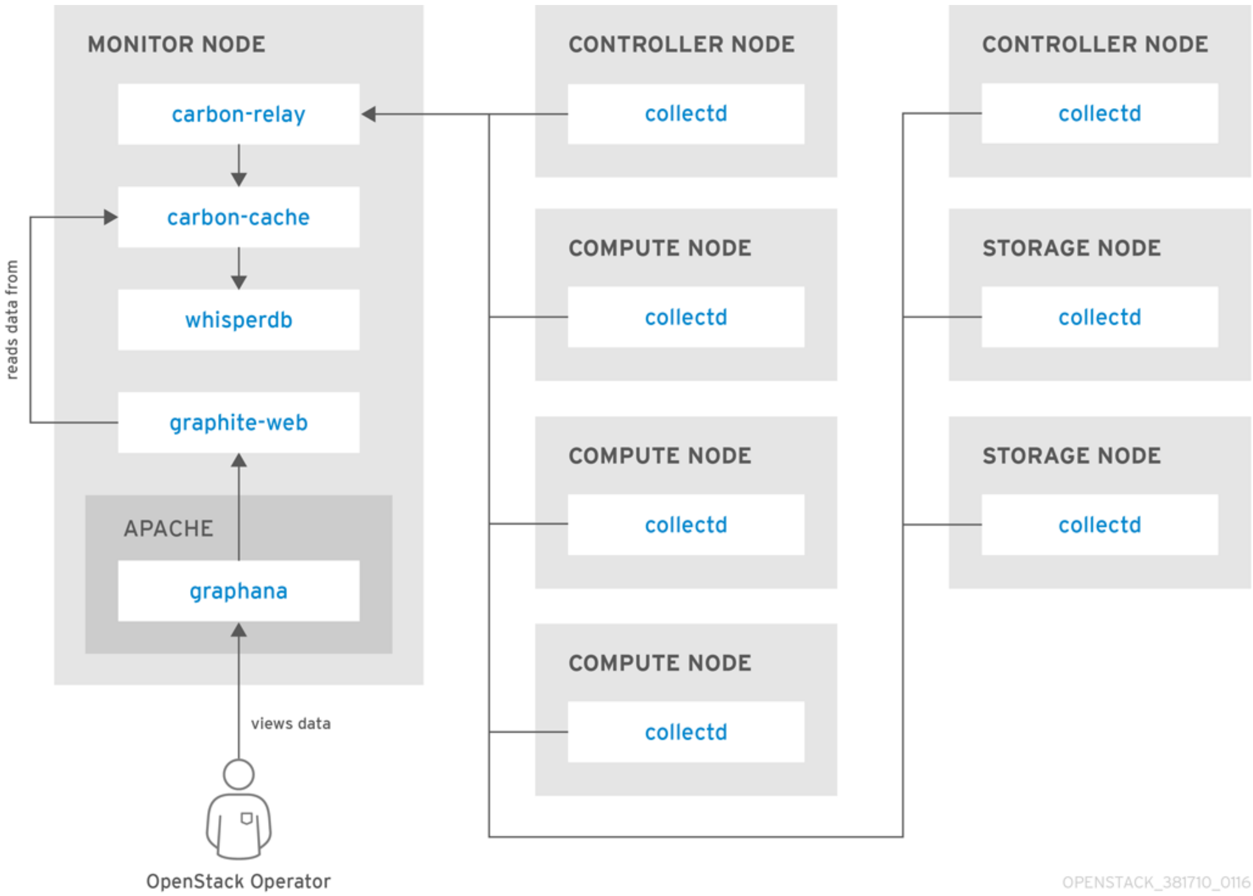
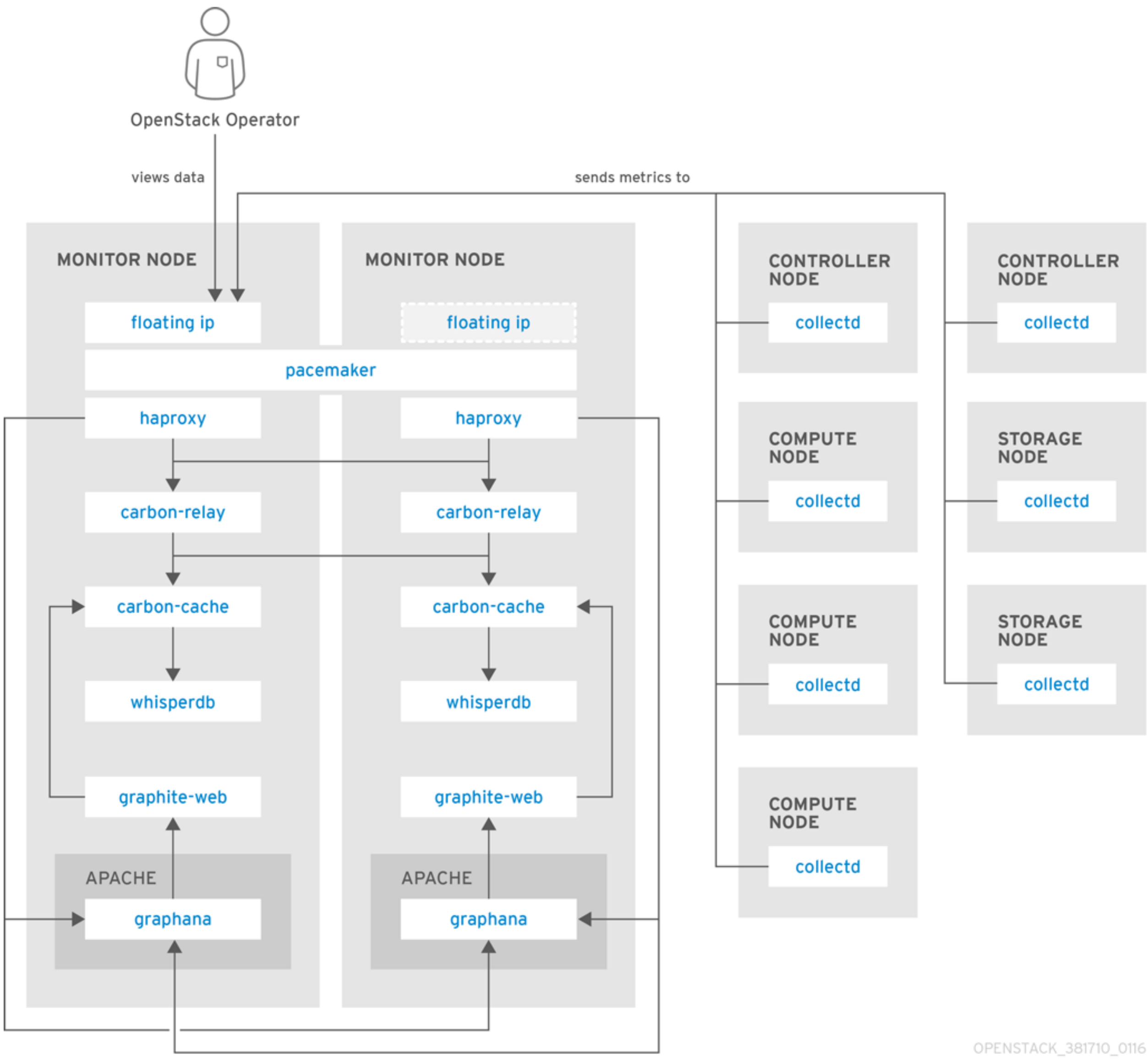


Figure 1.9. HA deployment for Red Hat OpenStack Platform



CHAPTER 2. INSTALLING THE CENTRALIZED LOGGING SUITE

2.1. INSTALLING THE CENTRALIZED LOG RELAY/TRANSFORMER

1. Locate a bare metal system that meets the following minimum specifications:
   - 8 GB of memory
   - Single-socket Xeon class CPU
   - 500 GB of disk space
2. Install Red Hat Enterprise Linux 7.
3. Allow the system to access the Operational Tools packages:
   1. Register the system and subscribe it:

      ```
      # subscription-manager register
      # subscription-manager list --consumed
      ```

      If an OpenStack subscription is not attached immediately, see the documentation for manually attaching subscriptions.

   2. Disable initially enabled repositories and enable only the ones appropriate for the Operational Tools:

      ```
      # subscription-manager repos --disable=*
      # subscription-manager repos --enable=rhel-7-server-rpms --
      enable=rhel-7-server-optional-rpms --enable=rhel-7-server-
      openstack-8-optools-rpms
      ```

      Note

      The base OpenStack repository (rhel-7-server-openstack-8-rpms) must not be enabled on this node. This repository may contain newer versions of certain Operational Tools dependencies which may be incompatible with the Operational Tools packages.

4. Install the **`Elasticsearch`**, **`Fluentd`**, and **`Kibana`** software by running the following command:

   ```
   # yum install elasticsearch fluentd rubygem-fluent-plugin-
   elasticsearch kibana httpd
   ```

5. Configure **`Elasticsearch`**. To do this, edit **`/etc/elasticsearch/elasticsearch.yml`** and add the following lines to the end of the file:

```
http.cors.enabled: true
http.cors.allow-origin: "/.*/"
```

6. Start the **Elasticsearch** instance and enable it at boot:

```
# systemctl start elasticsearch
# systemctl enable elasticsearch
```

To confirm the **Elasticsearch** instance is working, run the following command and confirm it returns a valid response similar to below:

```
# curl http://localhost:9200/
```

This should give the following response:

```
{
  "status" : 200,
  "name" : "elasticsearch.example.com",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "1.5.2",
    "build_hash" : "c88f77ffc81301dfa9dfd81ca2232f09588bd512",
    "build_timestamp" : "2015-02-19T13:05:36Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.3"
  },
  "tagline" : "You Know, for Search"
}
```

7. Configure **Fluentd** to accept log data and write it to**Elasticsearch**. Edit **/etc/fluentd/fluent.conf** and replace its content with the following:

```
# In v1 configuration, type and id are @ prefix parameters.
# @type and @id are recommended. type and id are still available
for backward compatibility

<source>
  @type forward
  port 4000
  bind 0.0.0.0
</source>

<match **>
  @type elasticsearch
  host localhost
  port 9200
  logstash_format true
  flush_interval 5
</match>
```

8. Start **Fluentd** and enable it at boot:

```
# systemctl start fluentd
# systemctl enable fluentd
```

Tip

Check the journal for **Fluentd** and ensure it has no errors at start:

```
# journalctl -u fluentd -l -f
```

9. Configure **Kibana** to point to the **Elasticsearch** instance. Create **/etc/httpd/conf.d/kibana3.conf** and place the following content inside:

```
<VirtualHost *:80>

  DocumentRoot /usr/share/kibana
  <Directory /usr/share/kibana>
    Require all granted
    Options -Multiviews
  </Directory>

  # Proxy for _aliases and .*/_search
  <LocationMatch
"^/(_nodes|_aliases|.*/_aliases|_search|.*/_search|_mapping|.*/_m
apping)$">
    ProxyPassMatch http://127.0.0.1:9200/$1
    ProxyPassReverse http://127.0.0.1:9200/$1
  </LocationMatch>

  # Proxy for kibana-int/{dashboard,temp}
  <LocationMatch "^/(kibana-int/dashboard/|kibana-int/temp)
(.*)$">
    ProxyPassMatch http://127.0.0.1:9200/$1$2
    ProxyPassReverse http://127.0.0.1:9200/$1$2
  </LocationMatch>

</VirtualHost>
```

10. Enable **Kibana** (inside Apache) to connect to **Elasticsearch**, and then start Apache and enable it at boot:

```
# setsebool -P httpd_can_network_connect 1
# systemctl start httpd
# systemctl enable httpd
```

11. Open the firewall on the system to allow connections to **Fluentd**, **Elasticsearch**, and **httpd**:

```
# firewall-cmd --zone=public --add-port=4000/tcp --permanent
# firewall-cmd --zone=public --add-port=9200/tcp --permanent
# firewall-cmd --zone=public --add-service=http --permanent
# firewall-cmd --reload
```

Important

The information provided by Kibana and Elasticsearch is available to anyone without any authentication. To restrict access to these services, ensure that the system or the open TCP ports (80, 4000, and 9200) are only accessible from trusted hosts.

2.2. INSTALLING THE LOG COLLECTION AGENT ON ALL NODES

To collect the logs from all the systems in the OpenStack environment and send them to your centralized logging server, run the following commands on all the OpenStack systems.

1. Enable the Operational Tools repository:

```
# subscription-manager repos --enable=rhel-7-server-openstack-8-optools-rpms
```

2. Install **`fluentd`** and **`rubygem-fluent-plugin-add`**:

```
# yum install fluentd rubygem-fluent-plugin-add
```

3. Configure the **`Fluentd`** user so it has permissions to read all the OpenStack log files. Do this by running the following command:

```
# for user in {keystone,nova,neutron,cinder,glance}; do usermod -a -G $user fluentd; done
```

   Note that you may get an error on some nodes about missing groups. This can be disregarded as not all the nodes run all the services.

4. Configure **`Fluentd`**. Make sure **`/etc/fluentd/fluent.conf`** looks like the following; be sure to replace *LOGGING_SERVER* with the host name or IP address of your centralized logging server configured above:

```
# In v1 configuration, type and id are @ prefix parameters.
# @type and @id are recommended. type and id are still available
for backward compatibility

# Nova compute
<source>
  @type tail
  path /var/log/nova/nova-compute.log
  tag nova.compute
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match nova.compute>
  type add
  <pair>
    service nova.compute
```

```
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Nova API
<source>
  @type tail
  path /var/log/nova/nova-api.log
  tag nova.api
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match nova.api>
  type add
  <pair>
    service nova.api
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Nova Cert
<source>
  @type tail
  path /var/log/nova/nova-cert.log
  tag nova.cert
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match nova.cert>
  type add
  <pair>
    service nova.cert
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Nova Conductor
<source>
  @type tail
  path /var/log/nova/nova-conductor.log
  tag nova.conductor
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
```

```
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match nova.conductor>
  type add
  <pair>
    service nova.conductor
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Nova Consoleauth
<source>
  @type tail
  path /var/log/nova/nova-consoleauth.log
  tag nova.consoleauth
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match nova.consoleauth>
  type add
  <pair>
    service nova.consoleauth
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Nova Scheduler
<source>
  @type tail
  path /var/log/nova/nova-scheduler.log
  tag nova.scheduler
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match nova.scheduler>
  type add
  <pair>
    service nova.scheduler
    hostname "#{Socket.gethostname}"
  </pair>
```

```
</match>

# Neutron Openvswitch Agent
<source>
  @type tail
  path /var/log/neutron/openvswitch-agent.log
  tag neutron.openvswitch
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match neutron.openvswitch>
  type add
  <pair>
    service neutron.openvswitch
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Neutron Server
<source>
  @type tail
  path /var/log/neutron/server.log
  tag neutron.server
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match neutron.server>
  type add
  <pair>
    service neutron.server
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Neutron DHCP Agent
<source>
  @type tail
  path /var/log/neutron/dhcp-agent.log
  tag neutron.dhcp
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>
```

```
<match neutron.dhcp>
  type add
  <pair>
    service neutron.dhcp
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Neutron L3 Agent
<source>
  @type tail
  path /var/log/neutron/l3-agent.log
  tag neutron.l3
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match neutron.l3>
  type add
  <pair>
    service neutron.l3
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Neutron Metadata Agent
<source>
  @type tail
  path /var/log/neutron/metadata-agent.log
  tag neutron.metadata
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match neutron.metadata>
  type add
  <pair>
    service neutron.metadata
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Keystone
<source>
  @type tail
  path /var/log/keystone/keystone.log
```

```
  tag keystone
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match keystone>
  type add
  <pair>
    service keystone
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Glance API
<source>
  @type tail
  path /var/log/glance/api.log
  tag glance.api
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match glance.api>
  type add
  <pair>
    service glance.api
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Glance Registry
<source>
  @type tail
  path /var/log/glance/registry.log
  tag glance.registry
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match glance.registry>
  type add
```

```
  <pair>
    service glance.registry
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Cinder API
<source>
  @type tail
  path /var/log/cinder/api.log
  tag cinder.api
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match cinder.api>
  type add
  <pair>
    service cinder.api
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Cinder Scheduler
<source>
  @type tail
  path /var/log/cinder/scheduler.log
  tag cinder.scheduler
  format multiline
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match cinder.scheduler>
  type add
  <pair>
    service cinder.scheduler
    hostname "#{Socket.gethostname}"
  </pair>
</match>

# Cinder Volume
<source>
  @type tail
  path /var/log/cinder/volume.log
  tag cinder.volume
  format multiline
```

```
  format_firstline /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?
<loglevel>[^ ]*) (?<class>[^ ]*) \[(?<context>.*)\] (?
<message>.*)/
  format /(?<time>[^ ]* [^ ]*) (?<pid>[^ ]*) (?<loglevel>[^ ]*)
(?<class>[^ ]*) \[(?<context>.*)\] (?<message>.*)/
  time_format %F %T.%L
</source>

<match cinder.volume>
  type add
  <pair>
    service cinder.volume
    hostname "#{Socket.gethostname}"
  </pair>
</match>

<match greped.**>
  @type forward
  heartbeat_type tcp
  <server>
    name LOGGING_SERVER
    host LOGGING_SERVER
    port 4000
  </server>
</match>
```

5. Now that **Fluentd** has been configured, start the **Fluentd** service and enable it at boot:

```
# systemctl start fluentd
# systemctl enable fluentd
```

You should now be able to access **Kibana** running at **http://*LOGGING_SERVER*/index.html#/dashboard/file/logstash.json** and see logs start to populate.

Note

By default, the front page of the logging server, **http://*LOGGING_SERVER*/**, is a **Kibana** welcome screen providing technical requirements and additional configuration information. If you want the logs to be available here, replace the **default.json** file in the **Kibana** application directory with **logstash.json**, but first create a backup copy of **default.json** in case you need this file again in the future:

```
# mv /usr/share/kibana/app/dashboards/default.json
/usr/share/kibana/app/dashboards/default.json.orig
# cp /usr/share/kibana/app/dashboards/logstash.json
/usr/share/kibana/app/dashboards/default.json
```

CHAPTER 3. INSTALLING THE AVAILABILITY MONITORING SUITE

3.1. INSTALLING THE MONITORING RELAY/CONTROLLER

1. Locate a bare metal system that meets the following minimum specifications:
   - 4 GB of memory
   - Single-socket Xeon class CPU
   - 100 GB of disk space
2. Install Red Hat Enterprise Linux 7.
3. Allow the system to access the Operational Tools packages:
   1. Register the system and subscribe it:

      ```
      # subscription-manager register
      # subscription-manager list --consumed
      ```

      If an OpenStack subscription is not attached immediately, see the documentation for manually attaching subscriptions.
   2. Disable initially enabled repositories and enable only the ones appropriate for the Operational Tools:

      ```
      # subscription-manager repos --disable=*
      # subscription-manager repos --enable=rhel-7-server-rpms --
      enable=rhel-7-server-optional-rpms --enable=rhel-7-server-
      openstack-8-optools-rpms
      ```

      Note

      The base OpenStack repository (rhel-7-server-openstack-8-rpms) must not be enabled on this node. This repository may contain newer versions of certain Operational Tools dependencies which may be incompatible with the Operational Tools packages.
4. Open the firewall on the system to allow connections to **`RabbitMQ`** and **`Uchiwa`**:

   ```
   # firewall-cmd --zone=public --add-port=5672/tcp --permanent
   # firewall-cmd --zone=public --add-port=3000/tcp --permanent
   # firewall-cmd --reload
   ```
5. Install the components needed for the monitoring server:

   ```
   # yum install sensu uchiwa redis rabbitmq-server
   ```

6. Configure **RabbitMQ** and **Redis**, which are the backbone services. Start both **Redis** and **RabbitMQ** and enable them at boot:

```
# systemctl start redis
# systemctl enable redis
# systemctl start rabbitmq-server
# systemctl enable rabbitmq-server
```

7. Configure a new **RabbitMQ** virtual host for **sensu**, with a user name and password combination that can access the host:

```
# rabbitmqctl add_vhost /sensu
# rabbitmqctl add_user sensu sensu
# rabbitmqctl set_permissions -p /sensu sensu ".*" ".*" ".*"
```

8. Now that the base services are running and configured, configure the **Sensu** monitoring server. Create **/etc/sensu/conf.d/rabbitmq.json** with the following contents:

```
{
  "rabbitmq": {
    "port": 5672,
    "host": "localhost",
    "user": "sensu",
    "password": "sensu",
    "vhost": "/sensu"
  }
}
```

9. Next, create **/etc/sensu/conf.d/redis.json** with the following contents:

```
{
  "redis": {
    "port": 6379,
    "host": "localhost"
  }
}
```

10. Finally, create **/etc/sensu/conf.d/api.json** with the following contents:

```
{
  "api": {
    "bind": "0.0.0.0",
    "port": 4567,
    "host": "localhost"
  }
}
```

11. Start and enable all **Sensu** services:

```
# systemctl start sensu-server
# systemctl enable sensu-server
# systemctl start sensu-api
# systemctl enable sensu-api
```

12. Configure **Uchiwa**, which is the web interface for **Sensu**. To do this, edit **/etc/uchiwa/uchiwa.json** and replace its default contents with the following:

```
{
  "sensu": [
    {
      "name": "Openstack",
      "host": "localhost",
      "port": 4567
    }
  ],
  "uchiwa": {
    "host": "0.0.0.0",
    "port": 3000,
    "refresh": 5
  }
}
```

13. Start and enable the **Uchiwa** web interface:

```
# systemctl start uchiwa
# systemctl enable uchiwa
```

3.2. INSTALLING THE AVAILABILITY MONITORING AGENT ON ALL NODES

To monitor all the systems in the OpenStack environment, run the following commands on all of them.

1. Enable the Operational Tools repository:

```
# subscription-manager repos --enable=rhel-7-server-openstack-8-optools-rpms
```

2. Install **Sensu**:

```
# yum install sensu
```

3. Configure the **Sensu** agent. Edit **/etc/sensu/conf.d/rabbitmq.json** to have the following content; remember to replace *MONITORING_SERVER* with the host name or IP address of your monitoring server configured in the previous section:

```
{
  "rabbitmq": {
    "port": 5672,
    "host": "MONITORING_SERVER",
    "user": "sensu",
    "password": "sensu",
    "vhost": "/sensu"
  }
}
```

4. Edit **`/etc/sensu/conf.d/client.json`** to include the following content; remember to replace *FQDN* with the host name of the machine, and *ADDRESS* with the public IP address of the machine:

```
{
  "client": {
    "name": "FQDN",
    "address": "ADDRESS",
    "subscriptions": [ "all" ]
  }
}
```

5. Finally, start and enable the **Sensu** client:

```
# systemctl start sensu-client
# systemctl enable sensu-client
```

You should now be able to access **Uchiwa** running at **`http://MONITORING_SERVER/:3000`**.

CHAPTER 4. INSTALLING THE PERFORMANCE MONITORING SUITE

4.1. INSTALLING THE COLLECTION AGGREGATOR/RELAY

1. Locate a bare metal system that meets the following minimum specifications:
   - 4 GB of memory
   - Single-socket Xeon class CPU
   - 500 GB of disk space
2. Install Red Hat Enterprise Linux 7.
3. Allow the system to access the Operational Tools packages:
   1. Register the system and subscribe it:

      ```
      # subscription-manager register
      # subscription-manager list --consumed
      ```

      If an OpenStack subscription is not attached immediately, see the documentation for manually attaching subscriptions.
   2. Disable initially enabled repositories and enable only the ones appropriate for the Operational Tools:

      ```
      # subscription-manager repos --disable=*
      # subscription-manager repos --enable=rhel-7-server-rpms --
      enable=rhel-7-server-optional-rpms --enable=rhel-7-server-
      openstack-8-optools-rpms
      ```

      Note

      The base OpenStack repository (rhel-7-server-openstack-8-rpms) must not be enabled on this node. This repository may contain newer versions of certain Operational Tools dependencies which may be incompatible with the Operational Tools packages.
4. Open the firewall on the system to allow connections to **`Graphite`** and **`Grafana`**:

   ```
   # firewall-cmd --zone=public --add-port=2003/tcp --permanent
   # firewall-cmd --zone=public --add-port=3000/tcp --permanent
   # firewall-cmd --reload
   ```
5. Once that is done, install the **`Graphite`** and **`Grafana`** software by running the following command:

   ```
   # yum install python-carbon graphite-web grafana httpd
   ```

6. Configure the **Grafana** web interface to allow access. Edit **/etc/httpd/conf.d/graphite-web.conf** and modify the **Require** line as follows:

```
...
<Directory "/usr/share/graphite/">
    <IfModule mod_authz_core.c>
        # Apache 2.4
        Require all granted
    </IfModule>
...
```

7. Synchronize the database behind **Graphite** web. Run the following command; when prompted if you want to create a super user, choose **no**:

```
# sudo -u apache /usr/bin/graphite-manage syncdb --noinput
```

8. Start and enable all the **Graphite** and **Grafana** services:

```
# systemctl start httpd
# systemctl enable httpd
# systemctl start carbon-cache
# systemctl enable carbon-cache
# systemctl start grafana-server
# systemctl enable grafana-server
```

9. Configure **Grafana** to talk to your **Graphite** instance:

   a. Go to **http://*PERFORMANCE_MONITORING_HOST*:3000/**. You should be presented with the **Grafana** login page.

   b. Enter the default credentials of **admin/admin** to log in to the system.

   c. After you are logged in, click on the **Grafana** logo in the top left corner of the screen, then choose *Data Sources*.

   d. At the top of the page, click *Add new*, and enter the following details:

Name	**graphite**
Default	**yes (select)**
Type	**Graphite**
Url	**http://localhost/**
Access	**proxy**

Basic Auth	**no (unselected)**

e. Finally, click the *Add* button at the bottom.

4.2. INSTALLING THE PERFORMANCE MONITORING COLLECTION AGENT ON ALL NODES

To monitor the performance of all the systems in the OpenStack environment, run the following commands on all of them.

1. Enable the Operational Tools repository:

```
# subscription-manager repos --enable=rhel-7-server-openstack-8-
optools-rpms
```

2. Install **collectd**:

```
# yum install collectd
```

3. Configure **collectd** to send the data to the performance monitoring aggregator/relay. To do so, create **/etc/collectd.d/10-write_graphite.conf** with the following contents, where *PERFORMANCE_MONITORING_HOST* is the host name or IP address of the host that was configured previously to be the performance monitoring aggregator/relay:

```
<LoadPlugin write_graphite>
  Globals false
</LoadPlugin>

<Plugin write_graphite>
  <Carbon>
    Host "PERFORMANCE_MONITORING_HOST"
    Port "2003"
    Prefix "collectd."
    EscapeCharacter "_"
    StoreRates true
    LogSendErrors true
    Protocol "tcp"
  </Carbon>
</Plugin>
```

4. Start and enable **collectd**:

```
# systemctl start collectd
# systemctl enable collectd
```

After a while, you should see metrics in the **Graphite** web user interface running at **http://*PERFORMANCE_MONITORING_HOST*:3000/**.