



Red Hat OpenStack Platform

8

Red Hat OpenStack Platform 運用 ツール

OpenStack 環境の集中ロギングと監視

OpenStack Team

OpenStack 環境の集中ロギングと監視

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、集中ロギング、可用性の監視、パフォーマンスの監視を提供する運用ツールのインストールと設定の方法を説明します。

目次

前書き	3
第1章 アーキテクチャー	4
1.1. 集中ロギング	4
1.2. 可用性の監視	5
1.3. パフォーマンスの監視	8
第2章 集中ロギングスイートのインストール	11
2.1. 集中ログリレー/トランスフォーマーのインストール	11
2.2. 全ノード上でのログ収集エージェントのインストール	13
第3章 可用性監視スイートのインストール	22
3.1. 監視リレー/コントローラーのインストール	22
3.2. 全ノード上での可用性監視エージェントのインストール	24
第4章 パフォーマンス監視スイートのインストール	26
4.1. 収集アグリゲーター/リレーのインストール	26
4.2. 全ノード上でのパフォーマンス監視収集エージェントのインストール	28

前書き

Red Hat OpenStack Platform は、オペレーターが OpenStack 環境を維持管理しやすいように設計されたオプションのツールセットです。これらのツールは、以下の機能を実行します。

- ✎ 集中ロギング
- ✎ 可用性の監視
- ✎ パフォーマンスの監視

本ガイドでは、これらのツールの準備とインストールの方法を説明します。

警告

Red Hat OpenStack Platform の運用ツールスイートは現在テクノロジープレビュー扱いとなっています。Red Hat のテクノロジープレビューの詳しい情報は「[テクノロジープレビュー機能のサポート範囲](#)」を参照してください。

第1章 アーキテクチャ

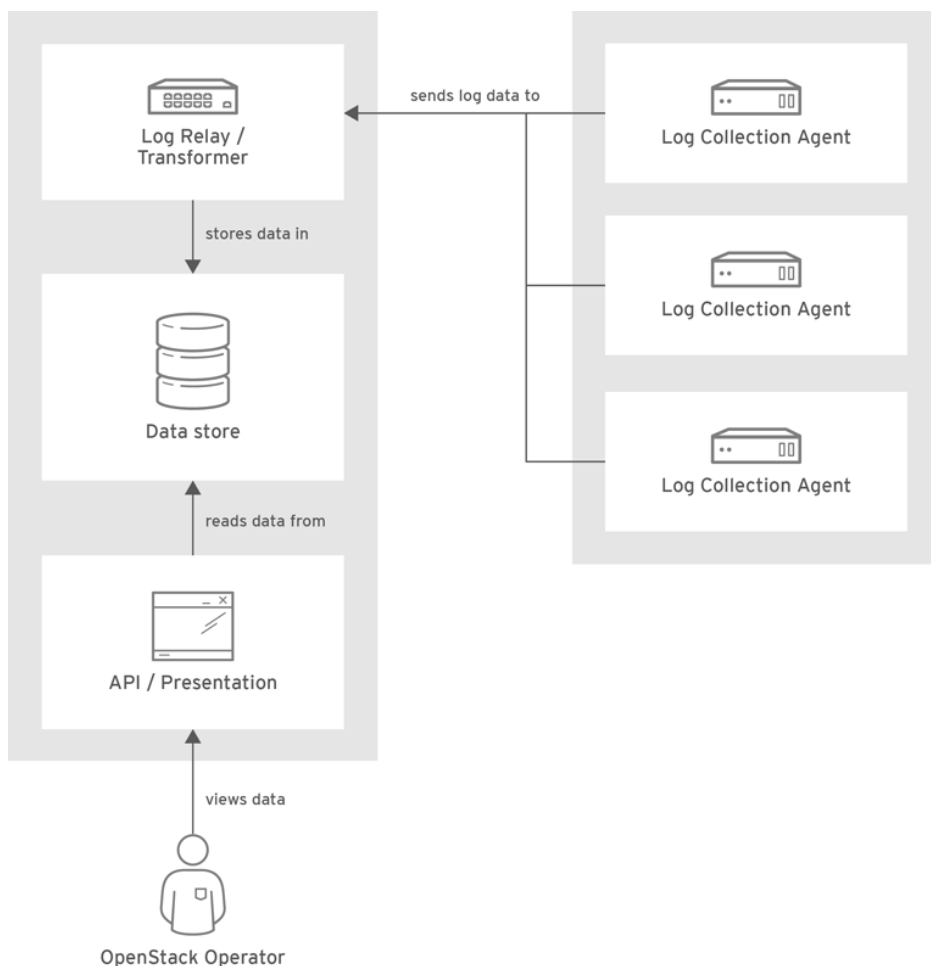
1.1. 集中ロギング

集中ロギングのツールチェーンは、以下のような複数のコンポーネントで構成されます。

- ※ ログ収集エージェント (Fluentd)
- ※ ログリレー/トランスフォーマー (Fluentd)
- ※ データストア (Elasticsearch)
- ※ API/プレゼンテーション層 (Kibana)

これらのコンポーネントとその対話については、以下の図に示されています。

図1.1 ハイレベルでの集中ロギングのアーキテクチャ



OPENSTACK_381710_0116

図1.2 Red Hat OpenStack Platform の単一ノードデプロイメント

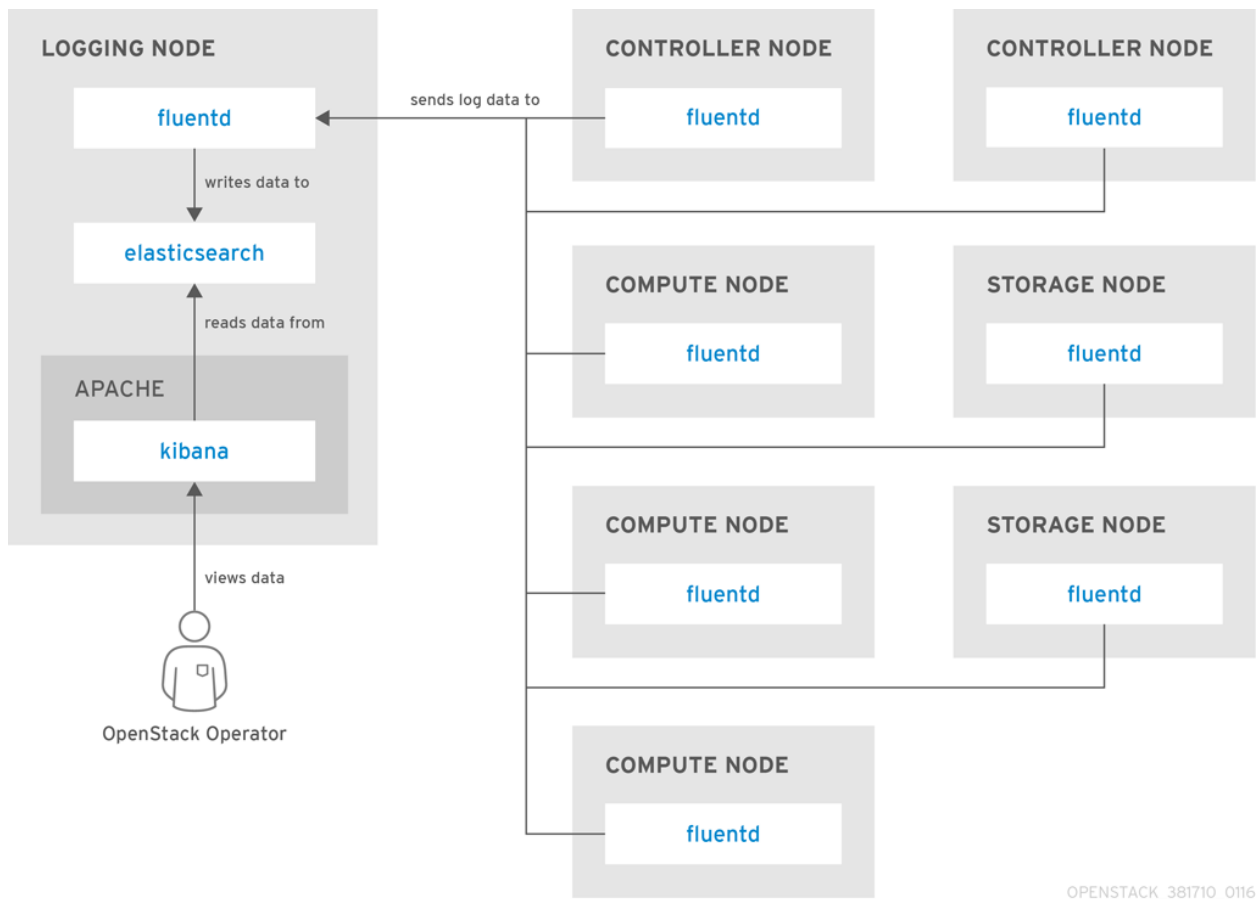
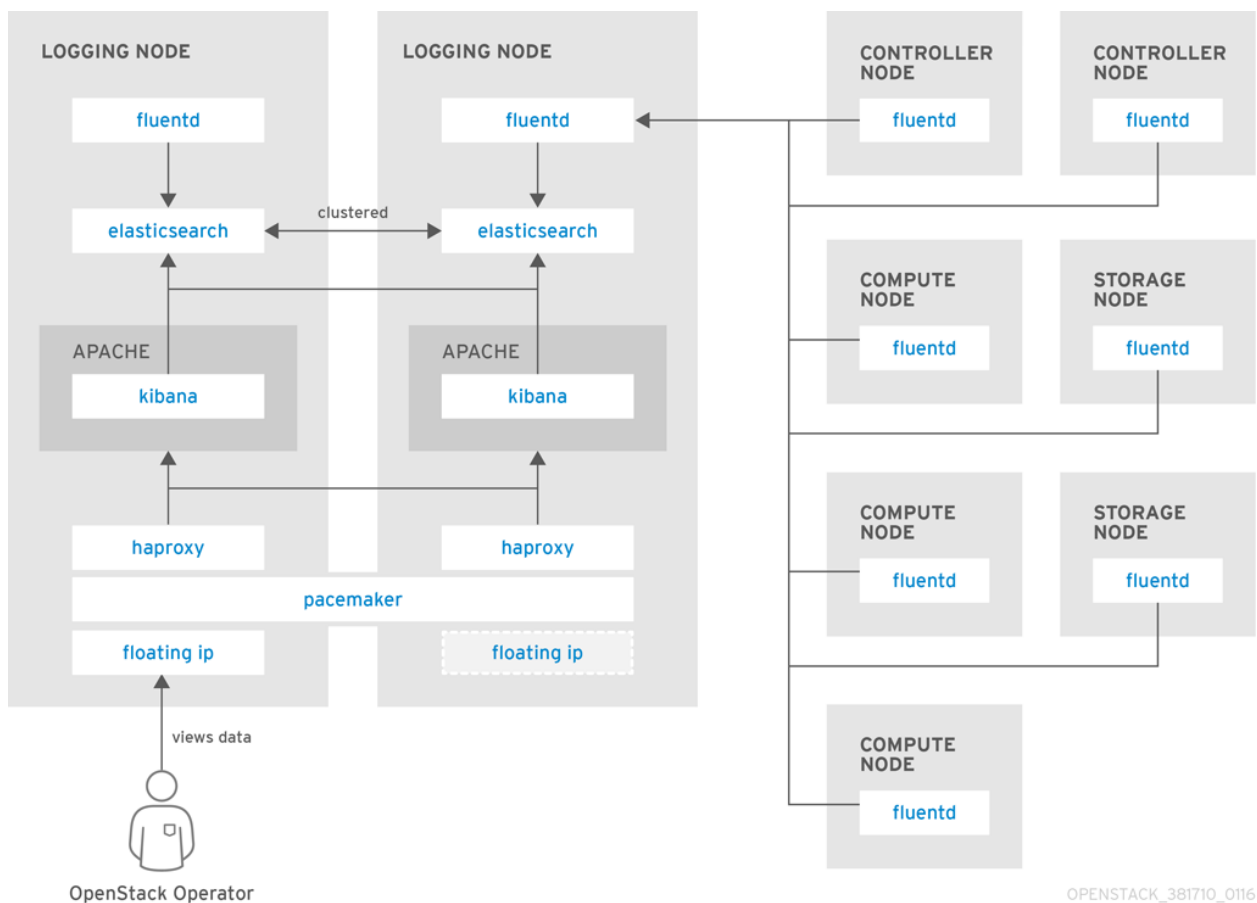


図1.3 RHOSP の HA デプロイメント



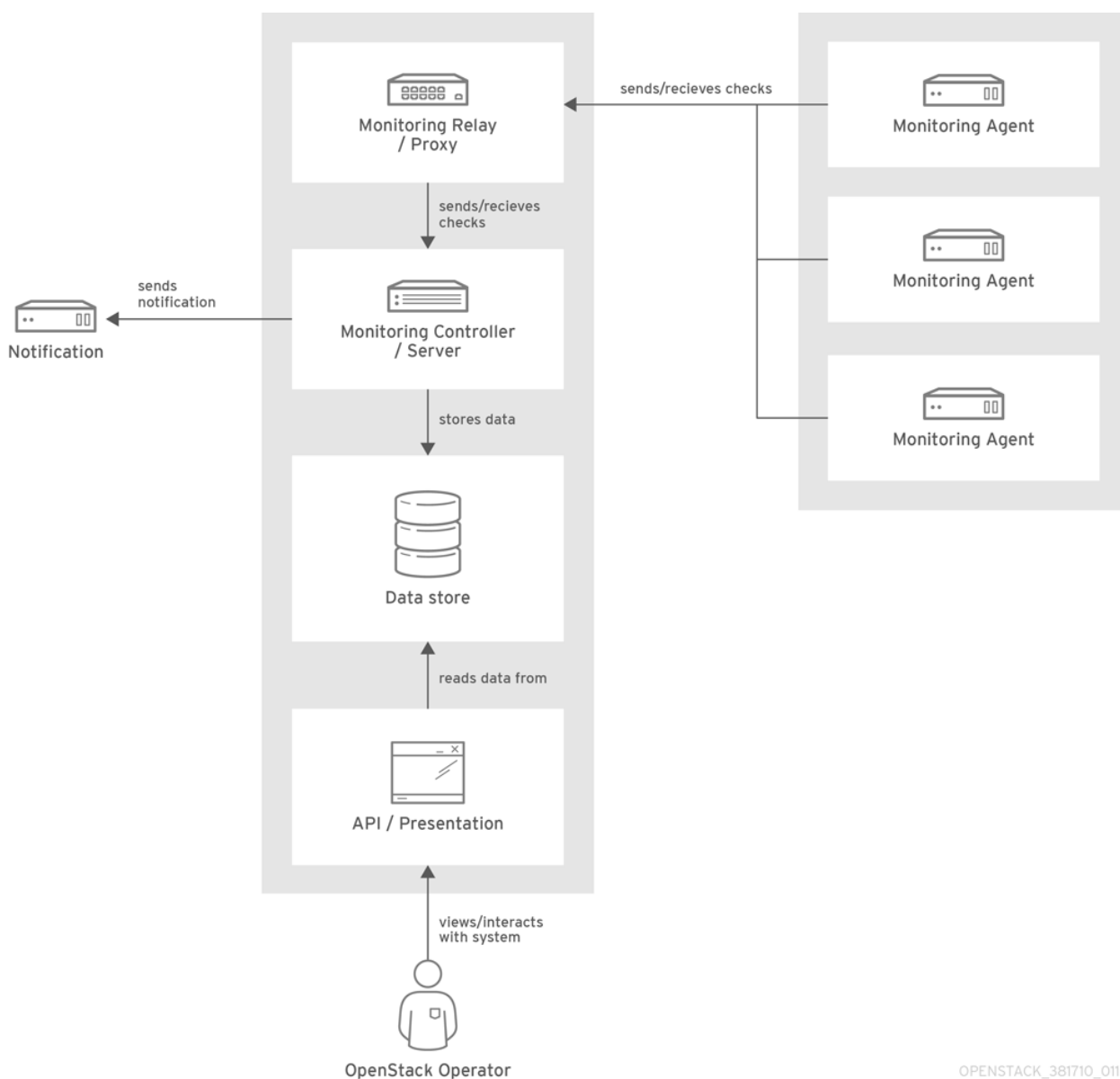
1.2.1 可用性監視

可用性監視のツールチェーンは、以下のような複数のコンポーネントで構成されます。

- ※ 監視エージェント (Sensu)
- ※ 監視リレー/プロキシ (RabbitMQ)
- ※ 監視コントローラー/サーバー (Sensu)
- ※ API/プレゼンテーション層 (Uchiwa)

これらのコンポーネントとその対話については、以下の図に示されています。

図1.4 ハイレベルでの可用性監視のアーキテクチャー



OPENSTACK_381710_0116

図1.5 Red Hat OpenStack Platform の単一ノードデプロイメント

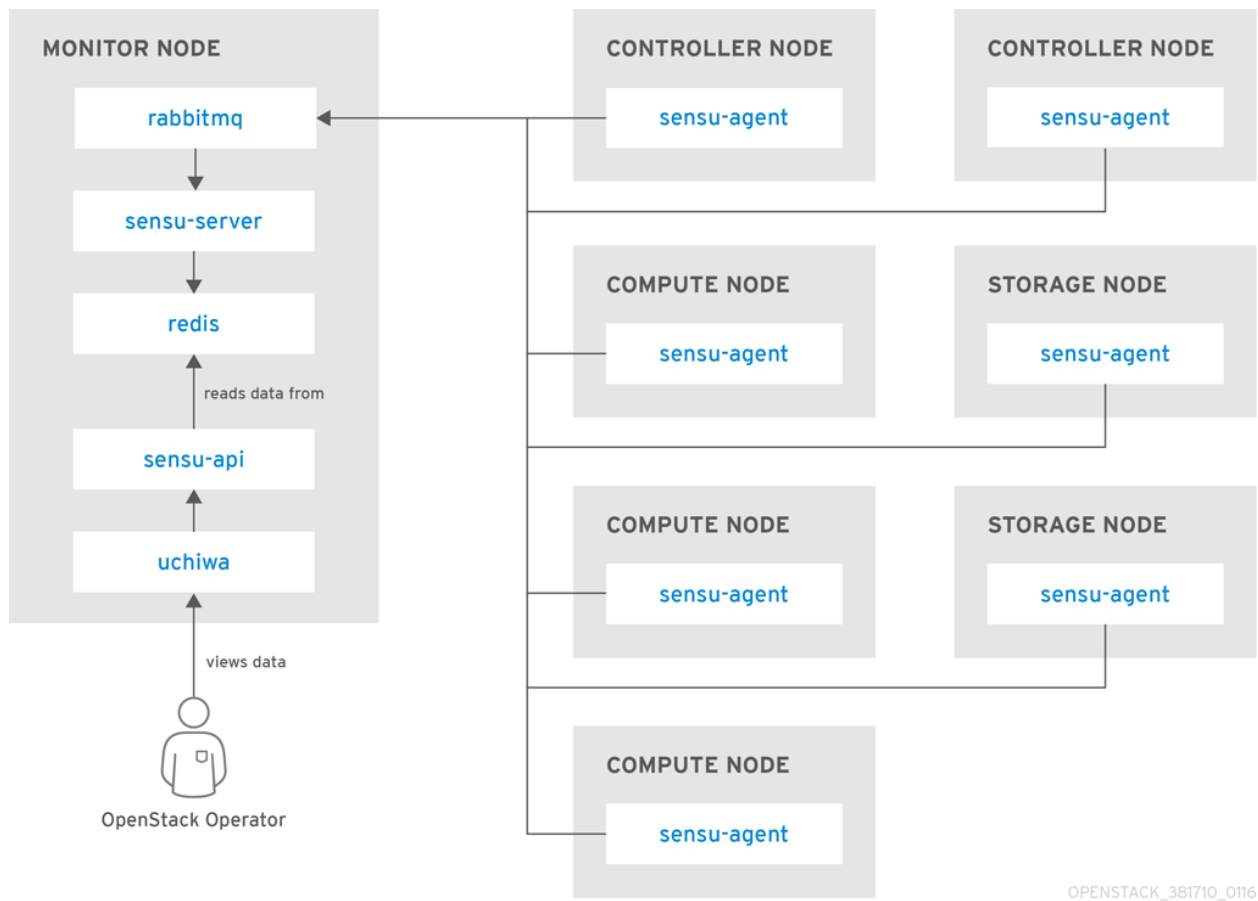
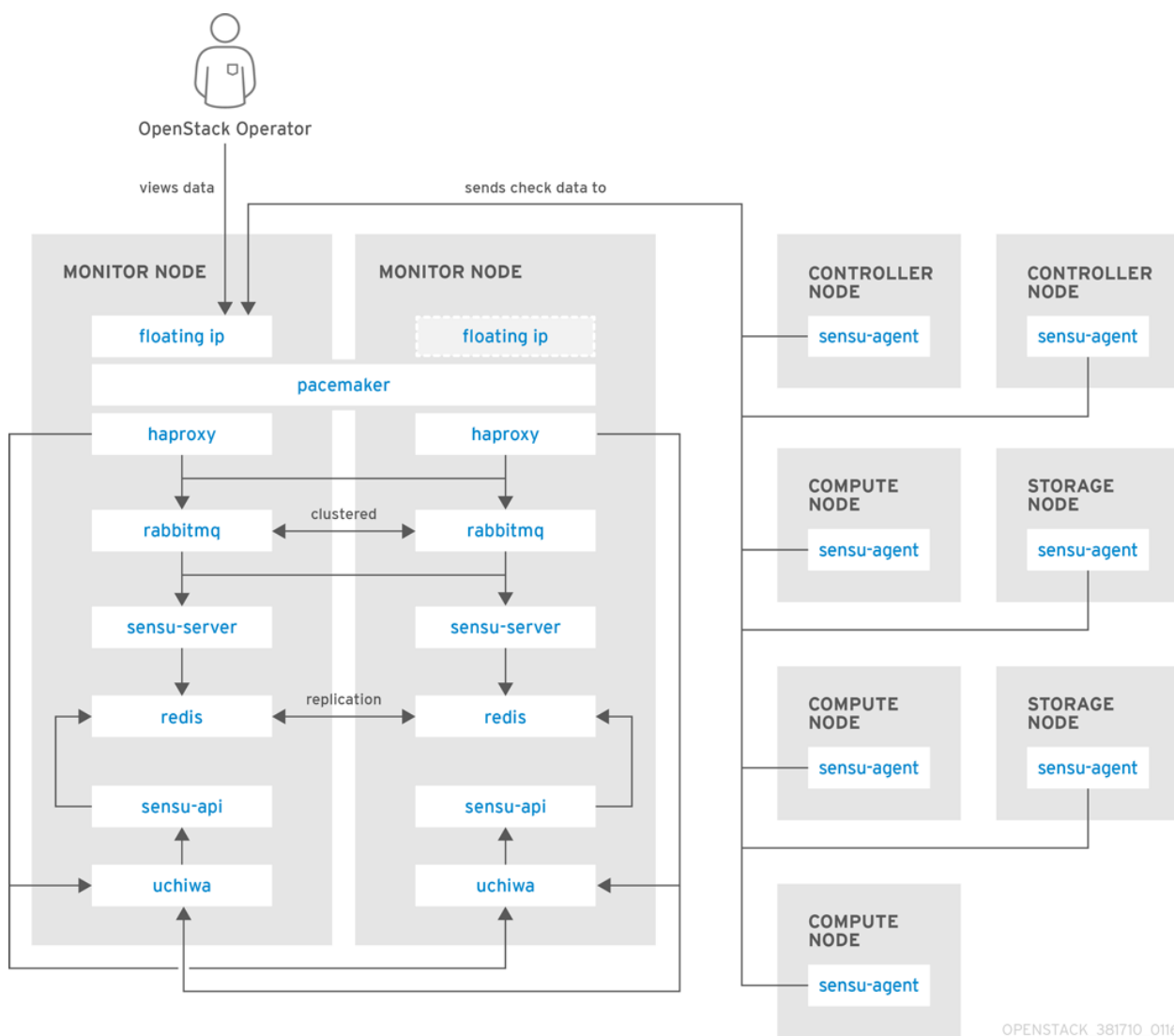


図1.6 RHOSP の HA デプロイメント



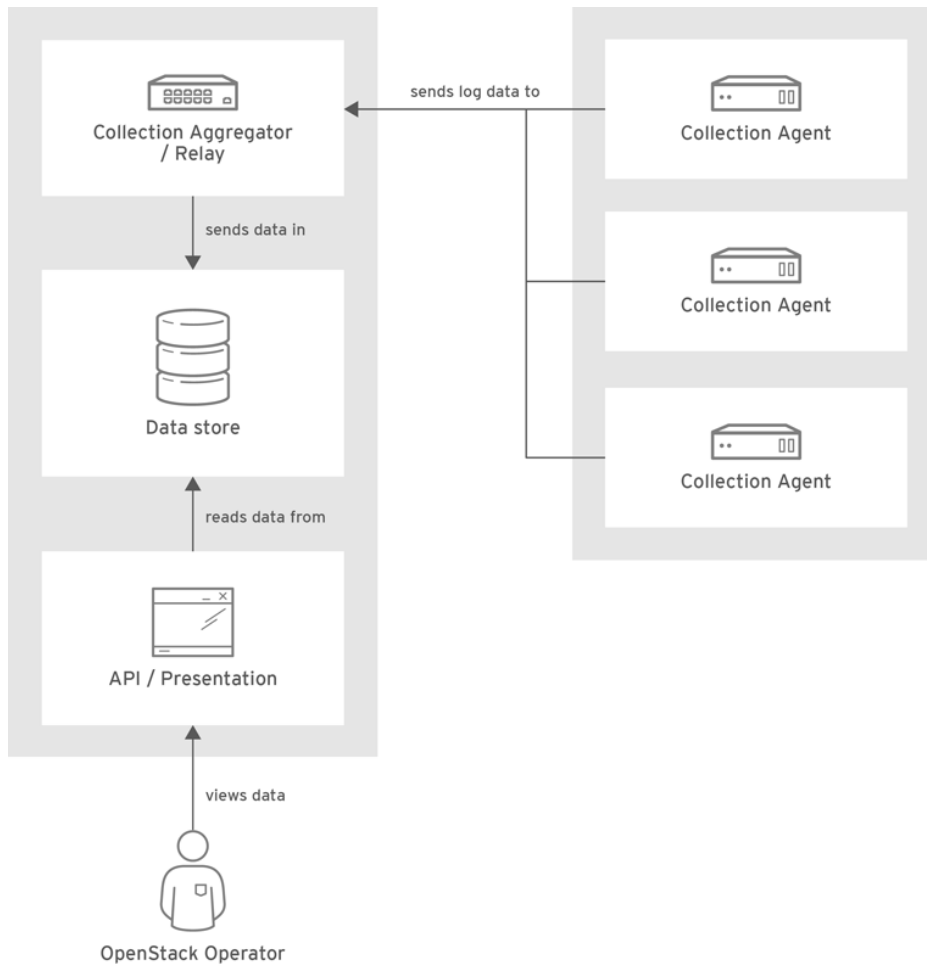
1.3. パフォーマンスの監視

パフォーマンス監視のツールチェーンは、以下のような複数のコンポーネントで構成されます。

- ※ 収集エージェント (collectd)
- ※ 収集アグリゲーター/リレー (Graphite)
- ※ データストア (whisperdb)
- ※ API/プレゼンテーション層 (Grafana)

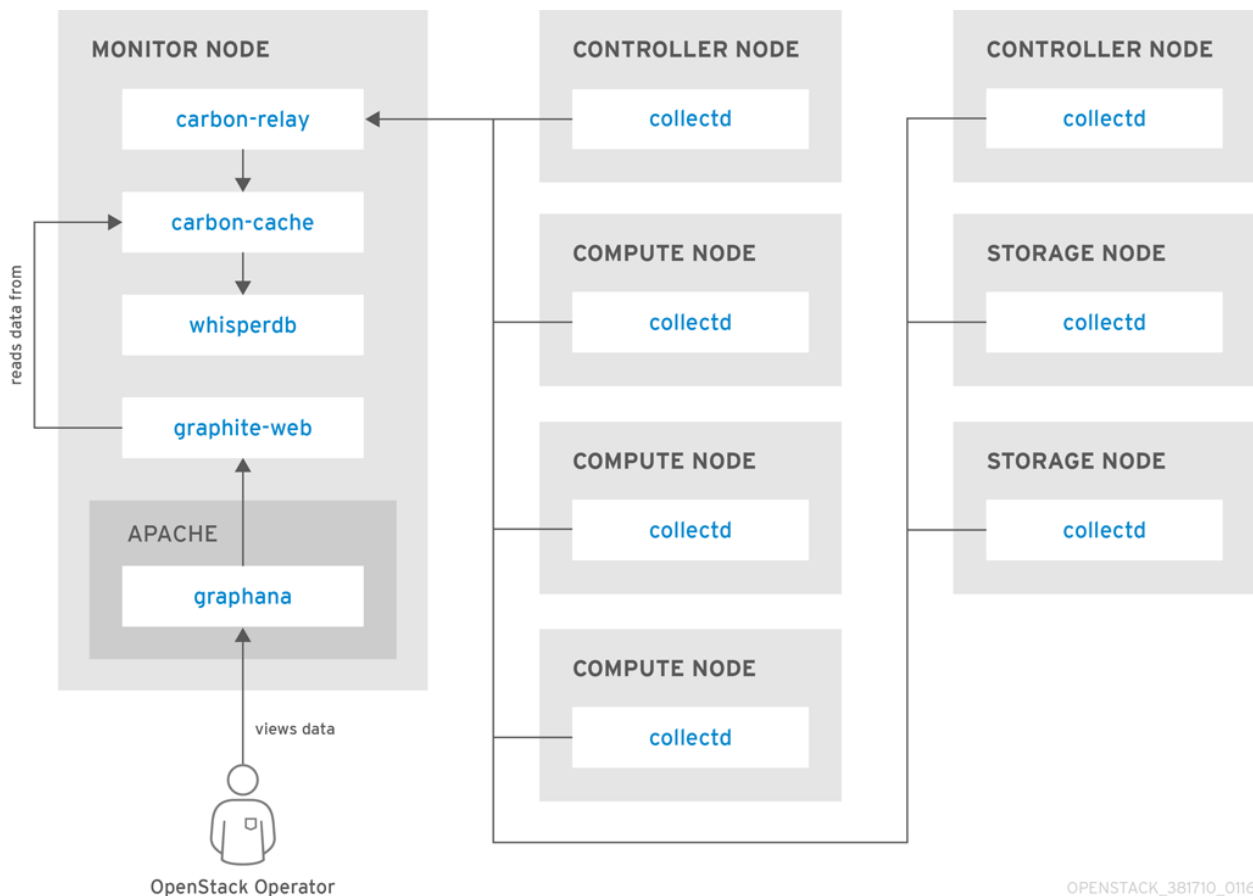
これらのコンポーネントとその対話については、以下の図に示されています。

図1.7 パフォーマンス監視のハイレベルアーキテクチャー



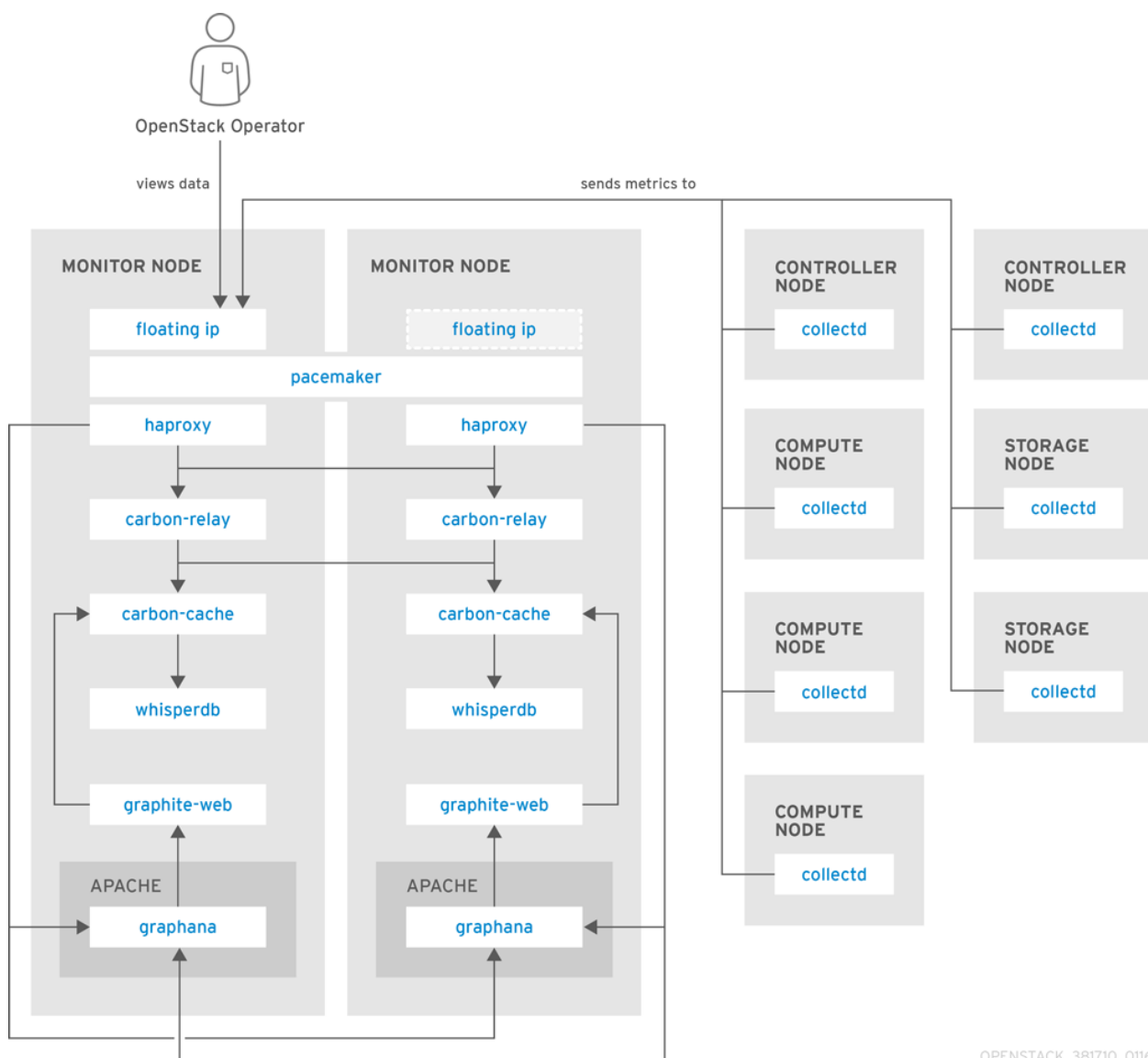
OPENSTACK_381710_0116

図1.8 Red Hat OpenStack Platform の単一ノードデプロイメント



OPENSTACK_381710_0116

図1.9 RHOSP の HA デプロイメント



OPENSTACK_381710_0116

第2章 集中ロギングスイートのインストール

2.1. 集中ログリレー/トランスフォーマーのインストール

1. 以下の最小仕様を満たすペアメタルのシステムを特定します。

- 8 GB メモリー
- 単一ソケットの Xeon クラスの CPU
- 500 GB のディスク領域

2. Red Hat Enterprise Linux 7 をインストールします。

3. このシステムの運用ツールパッケージへのアクセスを許可します。

1. システムを登録して、サブスクライブします。

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

OpenStack のサブスクリプションが自動的にアタッチされない場合は、[「manually attaching subscriptions」](#) に記載の説明を参照してください。

2. 最初に有効化されているリポジトリを無効化してから、運用ツールに適したリポジトリのみを有効化します。

```
# 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
```

注記

ベースの OpenStack リポジトリ (rhel-7-server-openstack-8-rpms) は、このノードでは有効化しないでください。このリポジトリには、特定の運用ツールの依存関係でより新しいバージョンが含まれている可能性があり、運用パッケージと互換性がない場合があります。

4. **Fluentd** と **httpd** に接続ができるようにシステムのファイアウォールを開放します。

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

5. 以下のコマンドで **Fluentd** と **Elasticsearch** のソフトウェアをインストールします。

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

6. **Elasticsearch** を設定します。これには、`/etc/elasticsearch/elasticsearch.yml` を編集して、ファイルの最後に以下

の行を追加します。

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

7. **Elasticsearch** インスタンスを起動して、ブート時に有効化されるように設定します。

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

Elasticsearch インスタンスが機能していることを確認するには、以下のコマンドを実行して、以下のように、有効な応答が返されることを確認します。

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

このコマンドを実行すると、以下のような応答が返されます。

```
{
  "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"
}
```

8. **Fluentd** がログデータを受け入れ、**Elasticsearch** に書き込みができるように設定します。`/etc/fluentd/fluent.conf` を編集して、内容を以下のように置き換えます。

```
# 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>
```

9. **Fluentd** を起動して、ブート時に有効化されるように設定します。


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

ヒント

Fluentd のジャーナルを確認して、起動時にエラーがないことを確認します。

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

10. **Kibana** が **Elasticsearch** インスタンスにポイントするように設定します。`/etc/httpd/conf.d/kibana3.conf` を作成して、このファイルに以下の内容を設定します。

```
<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|*/_mapping)$">
        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>
```

11. **Elasticsearch** に接続するように (Apache 内の) **Kibana** を有効化して、Apache を起動し、ブート時に有効化されるようにします。

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

2.2. 全ノード上でのログ収集エージェントのインストール

OpenStack 環境の全システムからのログを収集して集中ロギングサーバーに送信するには、全 OpenStack システムで以下のコマンドを実行します。

1. 運用ツールのリポジトリを有効にします。

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

2. **fluentd** と **rubygem-fluent-plugin-add** をインストールします。

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

3. **Fluentd** ユーザーがすべての OpenStack ログファイルを読み取りできるパーミッションが割り当てられるように設定します。これには、以下のコマンドを実行してください。

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

ノードによっては、グループがないとのエラーが発生する可能性がある点に注意してください。すべてのノードで全サービスが実行されるわけではないため、このエラーは無視しても構いません。

4. **Fluentd** を設定します。**/etc/fluentd/fluent.conf** が以下のようになるように設定します。**LOGGING_SERVER** は、上記で設定した集中ロギングサーバーのホスト名または IP アドレスに置き換えるのを忘れないでください。

```
# 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>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:
<loglevel>[^\s]*) (?:<class>[^\s]*) \[(?:<context>.*)\] (?:
<message>.*)/
  format /(?:<time>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:<loglevel>[^\s]*)
(?:<class>[^\s]*) \[(?:<context>.*)\] (?:<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>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:
<loglevel>[^\s]*) (?:<class>[^\s]*) \[(?:<context>.*)\] (?:
<message>.*)/
  format /(?:<time>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:<loglevel>[^\s]*)
(?:<class>[^\s]*) \[(?:<context>.*)\] (?:<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>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:
<loglevel>[^\s]*) (?:<class>[^\s]*) \[(?:<context>.*)\] (?:
<message>.*)/
  format /(?:<time>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:<loglevel>[^\s]*)
(?:<class>[^\s]*) \[(?:<context>.*)\] (?:<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>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:
<loglevel>[^\s]*) (?:<class>[^\s]*) \[(?:<context>.*)\] (?:
<message>.*)/
  format /(?:<time>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:<loglevel>[^\s]*)
(?:<class>[^\s]*) \[(?:<context>.*)\] (?:<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>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:
<loglevel>[^\s]*) (?:<class>[^\s]*) \[(?:<context>.*)\] (?:
<message>.*)/
  format /(?:<time>[^\s]* [^\s]*) (?:<pid>[^\s]*) (?:<loglevel>[^\s]*)

```

```

    (?<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. **Fluentd** が設定されたので、**Fluentd** サービスを起動して、ブート時に有効になるように設定します。

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

http://LOGGING_SERVER/index.html#/dashboard/file/logstash.json で実行中の **Kibana** にアクセスでき、ログの生成が開始されていることが分かるはずです。

注記

デフォルトでは、ロギングサーバーのフロントページ (**http://LOGGING_SERVER/**) は、技術要件と追加の設定情報を提供する **Kibana** のウェルカム画面になっています。ここにログを表示するには、**Kibana** アプリケーションディレクトリーの **default.json** ファイルを **logstash.json** に置き換えてください。ただし、後ほどこのファイルがもう一度必要になったときに備えて、まず **default.json** のバックアップコピーを作成してください。

```
# 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
```

第3章 可用性監視スイートのインストール

3.1. 監視リレー/コントローラーのインストール

1. 以下の最小仕様を満たすベアメタルのシステムを特定します。
 - ▶ 4 GB メモリー
 - ▶ 単一ソケットの Xeon クラスの CPU
 - ▶ 100 GB のディスク領域
2. Red Hat Enterprise Linux 7 をインストールします。
3. このシステムの運用ツールパッケージへのアクセスを許可します。

1. システムを登録して、サブスクライブします。

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

OpenStack のサブスクリプションが自動的にアタッチされない場合は、[「manually attaching subscriptions」](#) に記載の説明を参照してください。

2. 最初に有効化されているリポジトリを無効化してから、運用ツールに適したリポジトリのみを有効化します。

```
# 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
```

注記

ベースの OpenStack リポジトリ (rhel-7-server-openstack-8-rpms) は、このノードでは有効化しないでください。このリポジトリには、特定の運用ツールの依存関係でより新しいバージョンが含まれている可能性があり、運用パッケージと互換性がない場合があります。

4. **RabbitMQ** と **Uchiwa** に接続ができるようにシステムのファイアウォールを開放します。

```
# firewall-cmd --zone=public --add-port=5672/tcp --permanent
# firewall-cmd --zone=public --add-port=3000/tcp --permanent
# firewall-cmd --reload
```

5. 監視サーバーに必要なコンポーネントをインストールします。

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

6. 基幹サービスである **RabbitMQ** と **Redis** を設定します。**Redis** と **RabbitMQ** の両方を起動して、ブート時に有効になるように設定します。

■

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

7. **sensu** に新しい **RabbitMQ** 仮想ホストを設定して、このホストにアクセスできるユーザー名とパスワードの組み合わせを指定します。

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

8. ベースのサービスが稼動し、ベースサービスの設定が済みました。次に **Sensu** の監視サービスを設定します。以下の内容を設定した **/etc/sensu/conf.d/rabbitmq.json** を作成します。

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

9. 次に、以下の内容の **/etc/sensu/conf.d/redis.json** を作成します。

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

10. 最後に以下の内容の **/etc/sensu/conf.d/api.json** を作成します。

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

11. 全 **Sensu** サービスを起動して、有効化します。

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

12. **Sensu** の Web インターフェースである **Uchiwa** を設定します。これには、`/etc/uchiwa/uchiwa.json` を編集して、デフォルトの内容を以下に置き換えます。

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

13. **Uchiwa** の Web インターフェースを起動して、有効にします。

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

3.2. 全ノード上での可用性監視エージェントのインストール

OpenStack 環境のすべてのシステムを監視するには、環境内の全システムで以下のコマンドを実行します。

1. 運用ツールのリポジトリを有効にします。

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

2. **Sensu** をインストールします。

```
# yum install sensu
```

3. **Sensu** エージェントを設定します。以下の内容が含まれるように `/etc/sensu/conf.d/rabbitmq.json` を設定します。**MONITORING_SERVER** は、前のセクションで設定した監視サーバーのホスト名または IP アドレスに忘れずに置き換えるようにしてください。

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

4. 以下の内容が含まれるように `/etc/sensu/conf.d/client.json` を編集します。**FQDN** はマシンのホスト名に、**ADDRESS** はマシンのパブリック IP に忘れずに置き換えてください。

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

5. 最後に **Sensu** クライアントを起動して、有効化します。

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

`http://MONITORING_SERVER/:3000` で実行される **Uchiwa** にアクセスできるはずです。

第4章 パフォーマンス監視スイートのインストール

4.1. 収集アグリゲーター/リレーのインストール

1. 以下の最小仕様を満たすベアメタルのシステムを特定します。
 - 4 GB メモリー
 - 単一ソケットの Xeon クラスの CPU
 - 500 GB のディスク領域
2. Red Hat Enterprise Linux 7 をインストールします。
3. このシステムの運用ツールパッケージへのアクセスを許可します。

1. システムを登録して、サブスクライブします。

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

OpenStack のサブスクリプションが自動的にアタッチされない場合は、[「manually attaching subscriptions」](#) に記載の説明を参照してください。

2. 最初に有効化されているリポジトリを無効化してから、運用ツールに適したリポジトリのみを有効化します。

```
# 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
```

注記

ベースの OpenStack リポジトリ (rhel-7-server-openstack-8-rpms) は、このノードでは有効化しないでください。このリポジトリには、特定の運用ツールの依存関係でより新しいバージョンが含まれている可能性があり、運用パッケージと互換性がない場合があります。

4. **Graphite** および **Grafana** に接続ができるように、システムのファイアウォールを開放します。

```
# firewall-cmd --zone=public --add-port=2003/tcp --permanent
# firewall-cmd --zone=public --add-port=3000/tcp --permanent
# firewall-cmd --reload
```

5. ファイアウォールの開放が完了したら、以下のコマンドを実行して、**Graphite** と **Grafana** のソフトウェアをインストールします。

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

6. **Grafana** Web インターフェースへのアクセスを許可するように設定しま

す。`/etc/httpd/conf.d/graphite-web.conf` を編集して以下のように **Require** の行を変更してください。

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

```

7. **Graphite** Web の背後にあるデータベースを同期します。以下のコマンドを実行します。スーパーユーザーを作成するには、プロンプトの表示の際に **no** を選択します。

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

8. すべての **Graphite** と **Grafana** のサービスを起動して、有効化します。

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

```

9. **Grafana** が **Graphite** インスタンスと対話するように設定します。

- a. `http://PERFORMANCE_MONITORING_HOST:3000/` に移動すると、**Grafana** のログインページが表示されるはずです。
- b. デフォルトの認証情報 **admin/admin** を入力して、システムにログインします。
- c. ログイン後には、画面の左上部にある **Grafana** のロゴをクリックして、**Data Sources** を選択します。
- d. ページ上部の **Add new** をクリックして、以下の情報を入力します。

Name	graphite
Default	✓
Type	Graphite
Url	http://localhost/
Access	proxy

Basic Auth

選択なし

- e. 最後に一番下の **Add** ボタンをクリックします。

4.2. 全ノード上でのパフォーマンス監視収集エージェントのインストール

OpenStack 環境の全システムのパフォーマンスを監視するには、環境内の全システムで以下のコマンドを実行します。

1. 運用ツールのリポジトリを有効にします。

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

2. **collectd** をインストールします。

```
# yum install collectd
```

3. **collectd** がパフォーマンス監視アグリゲーター/リレーにデータを送信するように設定します。これには、以下の内容が含まれる **/etc/collectd.d/10-write_graphite.conf** を作成します。**PERFORMANCE_MONITORING_HOST** は、パフォーマンス監視アグリゲーター/リレーに先ほど設定したホストのホスト名または IP アドレスに置き換えてください。

```
<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. **collectd** を起動して、有効化します。

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

しばらくすると、**http://PERFORMANCE_MONITORING_HOST:3000/** で実行される **Graphite** Web ユーザーインターフェースにメトリックが表示されるはずです。

