



# Red Hat OpenStack Platform 13

## Red Hat OpenStack Platform の高可用性について の理解

Red Hat OpenStack Platform における高可用性の理解、デプロイ、管理



# Red Hat OpenStack Platform 13 Red Hat OpenStack Platform の高可用性 についての理解

---

Red Hat OpenStack Platform における高可用性の理解、デプロイ、管理

OpenStack Team  
rhos-docs@redhat.com

## 法律上の通知

Copyright © 2018 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

OpenStack の環境が効率的に稼働する状態を維持するには、Red Hat OpenStack Platform 13 director を使用して、OpenStack の主要な全サービスにわたって高可用性および負荷分散を提供する構成を作成できます。本ガイドでは以下の内容について説明します。Red Hat OpenStack Platform 13 director により作成された、基礎的な HA (高可用性) 環境。OpenStack HA 機能についての理解を高め、この機能を使用して作業を行うための参照モデルとして使用することができます。Red Hat OpenStack Platform 13 に搭載されているさまざまなサービスを高可用性にするために使用する HA 機能 Red Hat OpenStack Platform 13 の HA 機能を使用した作業およびトラブルシューティング用のツール例

## 目次

|  |           |
|--|-----------|
| <b>第1章 概要</b>  | <b>3</b>  |
| 1.1. 高可用性サービスの管理                                     | 3         |
| <b>第2章 RED HAT OPENSTACK PLATFORM の HA 機能についての理解</b> | <b>5</b>  |
| <b>第3章 OPENSTACK HA 環境へのログイン</b>                     | <b>6</b>  |
| <b>第4章 PACEMAKER の使用</b>                             | <b>8</b>  |
| 4.1. PACEMAKER の全般的な情報                               | 8         |
| 4.2. PACEMAKER で設定された仮想 IP アドレス                      | 9         |
| 4.3. PACEMAKER で設定された OPENSTACK サービス                 | 12        |
| 4.3.1. 簡易バンドルセットリソース (簡易バンドル)                        | 12        |
| 4.3.1.1. 簡易バンドルの設定                                   | 13        |
| 4.3.1.2. 簡易バンドルのステータスの確認                             | 14        |
| 4.3.2. 複合バンドルセットのリソース (複合バンドル)                       | 14        |
| 4.4. PACEMAKER の FAILED ACTIONS                      | 16        |
| 4.5. コントローラーのその他の PACEMAKER 情報                       | 16        |
| 4.6. フェンシング用のハードウェア                                  | 17        |
| <b>第5章 HAPROXY の使用</b>                               | <b>19</b> |
| 5.1. HAPROXY STATS                                   | 20        |
| 5.2. 参考資料  | 20        |
| <b>第6章 GALERA の使用</b>                                | <b>21</b> |
| 6.1. ホスト名の解決   | 21        |
| 6.2. データベースクラスターの整合性                                 | 22        |
| 6.3. データベースクラスターノード                                  | 23        |
| 6.4. データベースレプリケーションのパフォーマンス                          | 24        |
| <b>第7章 HA コントローラーリソースの調査と修正</b>                      | <b>27</b> |
| 7.1. コントローラー上のリソースの問題修正                              | 28        |
| <b>第8章 HA CEPH ノードの調査</b>                            | <b>31</b> |
| <b>付録A RED HAT OPENSTACK PLATFORM 13 HA 環境の構築</b>    | <b>33</b> |
| A.1. ハードウェアの仕様                                       | 33        |
| A.2. アンダークラウドの設定ファイル                                 | 35        |
| A.3. オーバークラウドの設定ファイル                                 | 38        |



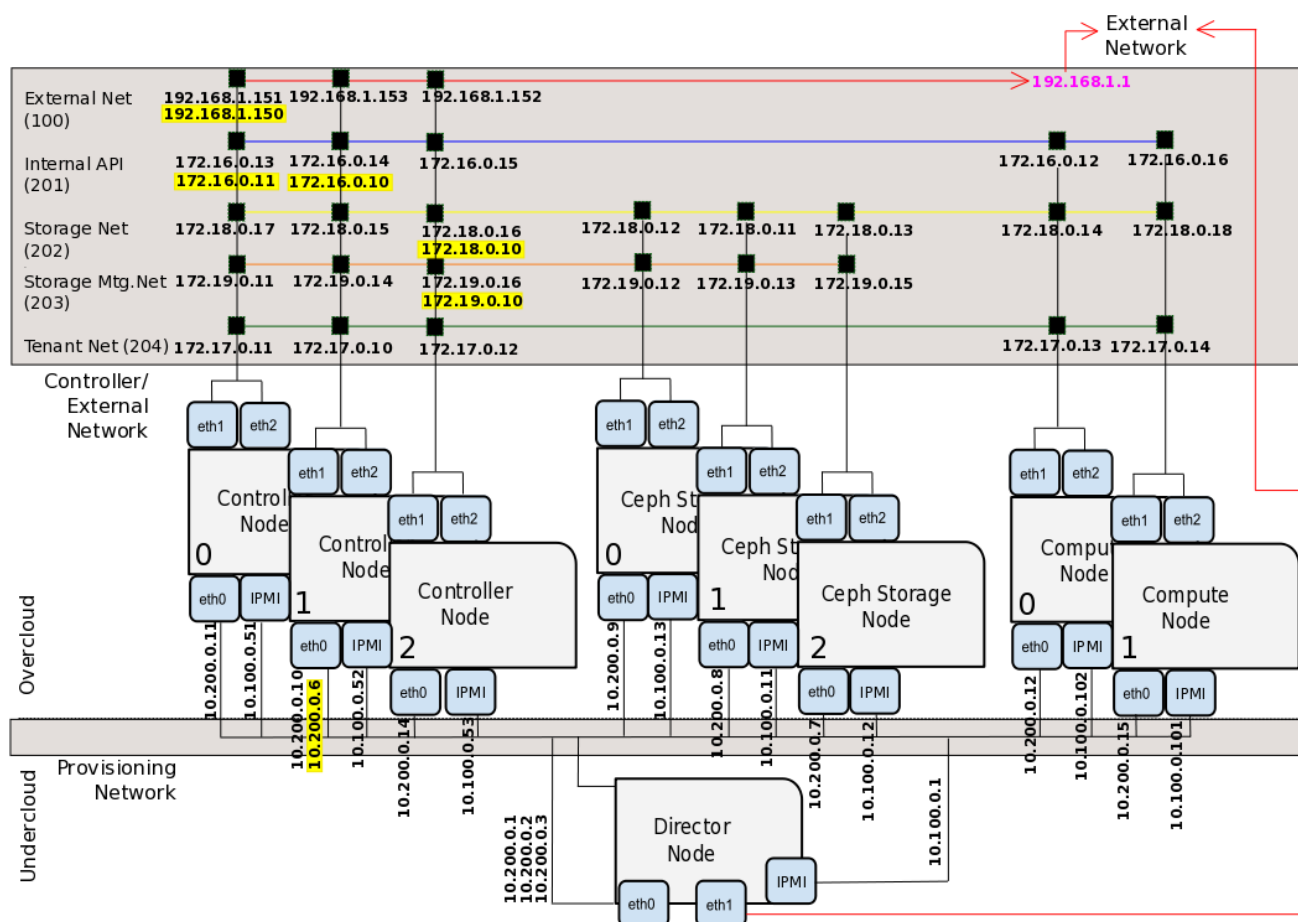
## 第1章 概要

本書で使用しているサンプルの HA デプロイメントは、以下のガイドを参考にしています。

- 『Deploying an Overcloud with Containerized Red Hat Ceph』
- 『director のインストールと使用方法』

下図は、本ガイドに記載の HA 機能のテスト専用で作成された特定の構成を示しています。この環境を再作成するための詳しい方法については、「[付録A Red Hat OpenStack Platform 13 HA 環境の構築](#)」の付録を参照して、手順を試すことができます。

図1.1 director を使用してデプロイした OpenStack HA 環境



### 1.1. 高可用性サービスの管理

高可用性 (HA) のデプロイメントでは、コアコンテナー、アクティブ/パッシブ、**systemd**、プレーンコンテナーの4つのタイプのサービスがあります。コアコンテナーとアクティブ/パッシブのサービスは Pacemaker によって起動/管理されます。その他のサービスはすべて、**systemd** で **systemctl** コマンドを使用するか、**Docker** で **docker** コマンドを使用して、直接管理されます。

#### コアコンテナー

コアコンテナサービスには、Galera、RabbitMQ、Redis、および HAProxyが含まれます。これらのサービスはすべてのコントローラード上で実行され、開始、停止、再起動の各処理に固有の管理と制約が必要です。

#### アクティブ/パッシブ

アクティブ/パッシブのサービスは、1回に1つのコントローラードでのみ実行され、**openstack-cinder-volume**などのサービスが含まれます。アクティブ/パッシブのサービス

の移動は Pacemaker で実行する必要があります。これにより、正しい停止-起動のシーケンスが確保されます。

### Systemd とブレーンコンテナ

Systemd およびブレーンコンテナのサービスは、独立したサービスで、サービスの中断に耐えることができる想定なので、Galera を再起動した場合に **neutron-server.service** や **openstack-nova-api-docker** などのコンテナのサービスを手動で再起動する必要はないはずです。

director を使用して HA デプロイメントを完全にオーケストレーションする場合には、director が使用するテンプレートと Puppet モジュールにより、HA 全サービスが設定され、特に HA に向けて正しく起動されます。また、HA の問題をトラブルシュートする場合には、HA フレームワーク、**docker** コマンド、**systemctl** コマンドのいずれかを使用してサービスと対話する必要があります。



## 第2章 RED HAT OPENSTACK PLATFORM の HA 機能についての理解

Red Hat OpenStack Platform は、高可用性を実装するために複数のテクノロジーを採用しています。高可用性は、OpenStack の設定でコントローラー、コンピューター、およびストレージノードを対象に異なる方法で提供されています。高可用性の実装方法を詳しく調べるには、各ノードにログインして、以下の項に記載したコマンドを実行してください。この操作の出力には、各ノードで HA サービスとプロセスが実行中であることが表示されます。

本ガイドに記載する高可用性 (HA) についての内容の大半は、コントローラーノードに関連しています。Red Hat OpenStack Platform のコントローラーノード上で使用されている主要な HA テクノロジーは 2 つあります。

- **Pacemaker:** Pacemaker は、仮想 IP アドレス、コンテナー、サービス、その他の機能をクラスター内のリソースとして設定することにより、定義済みの OpenStack クラスターリソースが確実に実行され、利用できるようにします。クラスター内のサービスノードまたは全ノードが停止した場合には、Pacemaker はリソースの再起動、クラスターからのノードの削除、ノードの再起動を実行することができます。これらのサービスの大半に対する要求は、HAProxy 経由で行われます。
- **HAProxy:** Red Hat OpenStack Platform で director を使用して複数のコントローラーノードを設定すると、HAProxy がこれらのノード上で実行中の OpenStack サービスの一部にトラフィックの負荷を分散するように設定されます。
- **Galera:** Red Hat OpenStack Platform は [MariaDB Galera Cluster](#) を使用して、データベースのレプリケーションを管理します。

OpenStack の高可用性サービスは、以下の 2 つのモードのいずれかで実行されます。

- **Active/Active:** このモードでは、Pacemaker により同じサービスが複数のノード上で起動し、トラフィックは HAProxy により要求サービスが実行中のノードに分散されるか、単一の IP アドレスを使用して特定のコントローラーに転送されます。場合によっては、HAProxy はトラフィックをラウンドロビン方式で Active/Active サービスに分散します。コントローラーノード数を増やすと、パフォーマンスを向上させることができます。
- **Active/passive:** Active/Active モードで実行できない、または Active/Active モードで実行するには信頼性に欠けるサービスは Active/Passive モードで実行します。これは、サービス内で 1 度にアクティブにできるインスタンスは 1 つのみという意味です。Galera の場合は、HAProxy は stick-table オプションを使用して、受信接続が単一のバックエンドサービスに転送されるようにします。サービスが複数の Galera ノードから同時に同一データにアクセスした場合には、Galera のマスター/マスターのモードはデッドロックになる可能性があります。

本ガイドに記載の高可用性サービスの使用を開始する際には、director システム (アンダークラウド) 自体も OpenStack を実行している点を念頭に置いてください。アンダークラウド (director システム) の目的は、実際に作業を行う OpenStack 環境のシステムを構築、管理することです。アンダークラウドから構築した環境は、オーバークラウドと呼ばれます。オーバークラウドを使用するには、本ガイドではアンダークラウドにログインしてから、調査するオーバークラウドノードを選択します。

## 第3章 OPENSTACK HA 環境へのログイン

OpenStack HA 環境が稼働している状態で、director (アンダークラウド) システムにログインしてから、以下のコマンドを実行して **stack** ユーザーになります。

```
# sudo su - stack
```

そこから、対応する環境変数を読み込んで、アンダークラウドまたはオーバークラウドと対話を行うことができます。

アンダークラウドと対話するには、以下のコマンドを実行します。

```
$ source ~/stackrc
```

オーバークラウドと対話するには、以下のコマンドを実行します。

```
$ source ~/overcloudrc
```

アンダークラウドまたはオーバークラウドへのアクセスに関する全般的な情報は、「[オーバークラウドへのアクセス](#)」のセクションを参照してください。

ノードにアクセスして調査するには、まず最初に、そのノードに割り当てられている IP アドレスを確認します。そのためには、アンダークラウドと対話する必要があります。

```
$ source ~/stackrc
$ openstack server list
+-----+-----+-----+-----+-----+
| ID      | Name                               | ... | Networks                               | ... |
+-----+-----+-----+-----+-----+
| d1...   | overcloud-controller-0            | ... | ctlplane=10.200.0.11 | ... |
...

```

## 注記

参考のために、本ガイドで使用しているテスト環境で director によってデプロイされたノードの名前とアドレスを以下の表にまとめます。

| 名前                      | IP アドレス            |
|-------------------------|--------------------|
| overcloud-controller-0  | <b>10.200.0.11</b> |
| overcloud-controller-1  | 10.200.0.10        |
| overcloud-controller-1  | 10.200.0.6 (仮想 IP) |
| overcloud-controller-2  | 10.200.0.14        |
| overcloud-compute-0     | 10.200.0.12        |
| overcloud-compute-1     | 10.200.0.15        |
| overcloud-cephstorage-0 | 10.200.0.9         |
| overcloud-cephstorage-1 | 10.200.0.8         |
| overcloud-cephstorage-2 | 10.200.0.7         |

お使いのテスト環境では、同じアドレス範囲を使用しても、各ノードに割り当てられる IP アドレスは異なる場合があります。

オーバークラウドノードの IP アドレスを確認したら、以下のコマンドを実行してそれらのノードの 1 つにログインすることができます。この操作を実行するには、オーバークラウドと対話する必要があります。たとえば、**overcloud-controller-0** に **heat-admin** ユーザーとしてログインするには、以下のコマンドを実行します。

```
$ ssh heat-admin@10.200.0.11
```

コントローラー、コンピューター、またはストレージシステムにログインした後は、そこで HA 機能についての調査を開始することができます。

## 第4章 PACEMAKER の使用

図1.1「[director を使用してデプロイした OpenStack HA 環境](#)」に記載した OpenStack 設定では、大半の OpenStack サービスは3つのコントローラーノードで実行されます。これらのサービスの高可用性機能を確認するには、**heat-admin** ユーザーとしてコントローラーのいずれかにログインして、Pacemaker が制御するサービスを確認します。

Pacemaker **pcs status** のコマンドの出力には、Pacemaker の全般的な情報、仮想 IP アドレス、サービス、および Pacemaker に関するその他の情報が含まれています。

Red Hat Enterprise Linux の Pacemaker に関する一般情報については、以下のリンクを参照してください。

- [『High Availability アドオンの概要』](#)
- [『High Availability Add-On の管理』](#)
- [『High Availability Add-On リファレンス』](#)

### 4.1. PACEMAKER の全般的な情報

以下の例は、**pcs status** コマンドで出力される Pacemaker の全般的な情報のセクションを示しています。

```
$ sudo pcs status
  Cluster name: tripleo_cluster ❶
  Stack: corosync
  Current DC: overcloud-controller-1 (version 1.1.16-12.el7_4.5-94ff4df)
  - partition with quorum

  Last updated: Thu Feb  8 14:29:21 2018
  Last change: Sat Feb  3 11:37:17 2018 by root via cibadmin on
overcloud-controller-2

  12 nodes configured ❷
  37 resources configured ❸

  Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ] ❹
  GuestOnline: [ galera-bundle-0@overcloud-controller-0 galera-bundle-
1@overcloud-controller-1 galera-bundle-2@overcloud-controller-2 rabbitmq-
bundle-0@overcloud-controller-0 rabbitmq-bundle-1@overcloud-controller-1
rabbitmq-bundle-2@overcloud-controller-2 redis-bundle-0@overcloud-
controller-0 redis-bundle-1@overcloud-controller-1 redis-bundle-
2@overcloud-controller-2 ] ❺

  Full list of resources:
[...]
```

この出力の主要なセクションでは、クラスターに関する以下の情報が表示されます。

- ❶ クラスター名
- ❷ クラスターを構成するノードの数

- 3 クラスターに設定されているリソースの数
- 4 現在オンライン状態のコントローラーノードの名前
- 5 現在オンライン状態のゲストノードの名前。各ゲストノードは、複合バンドルセットのリソースで構成されています。バンドルセットの詳細については、「[Pacemaker で設定された OpenStack サービス](#)」を参照してください。

## 4.2. PACEMAKER で設定された仮想 IP アドレス

各 IPAddr2 リソースは、クライアントがサービスへのアクセスを要求するために使用する仮想 IP アドレスを設定します。その IP アドレスに割り当てられているコントローラーノードでエラーが発生すると、IP アドレスは別のコントローラーに再割り当てされます。

この例では、特定の仮想 IP アドレスをリッスンするように現在設定されている各コントローラーノードを確認できます。

```
ip-10.200.0.6 (ocf::heartbeat:IPAddr2): Started overcloud-controller-1
ip-192.168.1.150 (ocf::heartbeat:IPAddr2): Started overcloud-controller-0
ip-172.16.0.10 (ocf::heartbeat:IPAddr2): Started overcloud-controller-1
ip-172.16.0.11 (ocf::heartbeat:IPAddr2): Started overcloud-controller-0
ip-172.18.0.10 (ocf::heartbeat:IPAddr2): Started overcloud-controller-2
ip-172.19.0.10 (ocf::heartbeat:IPAddr2): Started overcloud-controller-2
```

この出力では、各 IP アドレスが最初に特定のコントローラーに接続されます。たとえば、**192.168.1.150** は **overcloud-controller-0** で開始されます。ただし、そのコントローラーでエラーが発生すると、IP アドレスはクラスター内の他のコントローラーに再割り当てされます。

以下の表には、この例の IP アドレスと、各アドレスが最初に割り当てられた方法をまとめています。

表4.1 IP アドレスの説明と割り当て元

| IP アドレス              | 説明                                       | 割り当て元  |
|----------------------|--|--|
| <b>192.168.1.150</b> | パブリック IP アドレス                            | <b>network-environment.yaml</b> ファイルの <b>ExternalAllocationPools</b> 属性  |
| <b>10.200.0.6</b>    | コントローラーの仮想 IP アドレス                       | <b>dhcp_start</b> および <b>dhcp_end</b> の範囲の部分は、 <b>undercloud.conf</b> ファイルで <b>10.200.0.5-10.200.0.24</b> に設定されます。 |
| <b>172.16.0.10</b>   | コントローラー上の OpenStack API サービスへのアクセスを提供します | <b>network-environment.yaml</b> ファイルの <b>InternalApiAllocationPools</b>  |

| IP アドレス            | 説明  | 割り当て元  |
|--------------------|---|--|
| <b>172.18.0.10</b> | Glance API および Swift プロキシのサービスへのアクセスを提供するストレージの仮想 IP アドレス | <b>network-environment.yaml</b> ファイルの<br><b>StorageAllocationPools</b> 属性  |
| <b>172.16.0.11</b> | コントローラー上の Redis サービスへのアクセスを提供します                          | <b>network-environment.yaml</b> ファイルの<br><b>InternalApiAllocationPools</b> |
| <b>172.19.0.10</b> | ストレージ管理へのアクセスを提供します。                                      | <b>network-environment.yaml</b> ファイルの<br><b>StorageMgmtAllocationPools</b> |

**pcs** コマンドを使用して Pacemaker によって管理される特定の IP アドレスについての情報を確認することができます。たとえば、タイムアウトの情報やネットマスク ID を確認することができます。

以下の例は、**ip-192.168.1.150** のパブリック IP アドレスでコマンドを実行した場合の **pcs** コマンドの出力を示しています。

```
$ sudo pcs resource show ip-192.168.1.150
Resource: ip-192.168.1.150 (class=ocf provider=heartbeat type=IPaddr2)
Attributes: ip=192.168.1.150 cidr_netmask=32
Operations: start interval=0s timeout=20s (ip-192.168.1.150-start-timeout-20s)
              stop interval=0s timeout=20s (ip-192.168.1.150-stop-timeout-20s)
              monitor interval=10s timeout=20s (ip-192.168.1.150-monitor-interval-10s)
```

現在、192.168.1.150 のアドレスをリッスンするように割り当てられたコントローラーにログインしている場合には、以下のコマンドを実行して、コントローラーがアクティブな状態であることと、各種サービスがそのアドレスをアクティブにリッスンしていることを確認することができます。

```
$ ip addr show vlan100
9: vlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether be:ab:aa:37:34:e7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.151/24 brd 192.168.1.255 scope global vlan100
        valid_lft forever preferred_lft forever
    inet 192.168.1.150/32 brd 192.168.1.255 scope global vlan100
        valid_lft forever preferred_lft forever

$ sudo netstat -tupln | grep "192.168.1.150.*haproxy"
tcp        0      0 192.168.1.150:8778      0.0.0.0:*
LISTEN     61029/haproxy
tcp        0      0 192.168.1.150:8042      0.0.0.0:*
LISTEN     61029/haproxy
tcp        0      0 192.168.1.150:9292      0.0.0.0:*
LISTEN     61029/haproxy
```

```

tcp      0      0 192.168.1.150:8080      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:80        0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:8977      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:6080      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:9696      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:8000      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:8004      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:8774      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:5000      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:8776      0.0.0.0:*
LISTEN   61029/haproxy
tcp      0      0 192.168.1.150:8041      0.0.0.0:*
LISTEN   61029/haproxy

```

**ip** コマンドの出力には、**vlan100** のインターフェースが **192.168.1.150** と **192.168.1.151** の両方の IPv4 アドレスをリッスンしていることが示されています。

**netstat** コマンドの出力には、**192.168.1.150** インタフェースをリッスンしているすべてのプロセスが表示されています。ポート 123 でリッスンしている **ntpd** プロセスに加えて、**192.168.1.150** を特にリッスンしているその他唯一のプロセスは **haproxy** です。

#### 注記

**0.0.0.0**のように、すべてのローカルアドレスをリッスンしているプロセスは、192.168.1.150 から利用できます。これらのプロセスには、**sshd**、**mysqld**、**dhclient**、**ntpd** などがあります。

**netstat** コマンドの出力に表示されるポート番号は、HAProxy がリッスンしている特定のサービスを識別するのに役立ちます。これらのポート番号が表しているサービスは、**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ファイルで確認できます。

以下の一覧には、ポート番号とそれらに割り当てられたデフォルトのサービスの例をいくつか示しています。

- TCP ポート 6080: **nova\_novncproxy**
- TCP ポート 9696: **neutron**
- TCP ポート 8000: **heat\_cfn**
- TCP ポート 80: **horizon**
- TCP ポート 8776: **cinder**

現在、**haproxy.cfg** ファイルで定義されているサービスの大半は、3 つのコントローラーすべてで **192.168.1.150** の IP アドレスをリッスンしています。ただし、**192.168.1.150** の IP アドレスを外部でリッスンしているのは **controller-0** ノードのみです。

このため、**controller-0** ノードでエラーが発生した場合には、HAProxy は 192.168.1.150 を別のコントローラーに再割り当てするだけで、他のサービスはすべてフォールバックコントローラーノードですでに実行されている状態となります。

### 4.3. PACEMAKER で設定された OPENSTACK サービス

Red Hat OpenStack Platform 12 以降のクラスターで管理されるサービスの大半は、**バンドルセット** リソースまたは **バンドル** として設定されています。これらのサービスは、各コントローラーノードで同じ方法で開始することができ、常に各コントローラーで実行するように設定されます。

#### バンドル

すべてのコントローラーノードで同じ **コンテナ** を設定および複製し、必要なストレージパスをコンテナディレクトリーにマッピングして、リソース自体に関連する特定の属性を設定する **バンドル** リソース

#### コンテナ

コンテナは、haproxy のような単純なシステムベースのサービスから、異なるノード上のサービスの状態を制御および設定する特定のリソースエージェントを必要とする Galera のような複雑なサービスまで、さまざまな種類のリソースを実行できます。



#### 警告

- バンドルまたはコンテナを管理するための **docker** または **systemctl** コマンドの使用はサポートされていません。これらのコマンドは、サービスのステータスを確認に使用できますが、これらのサービスに対してアクションを実行するには Pacemaker のみを使用する必要があります。
- Pacemaker によって制御される Docker コンテナでは、**RestartPolicy** が Docker によって **no** に設定されます。これは、docker デーモンではなく、Pacemaker がコンテナの起動と停止のアクションを制御するようにするためです。

#### 4.3.1. 簡易バンドルセットリソース (簡易バンドル)

簡易バンドルセットリソースまたは **簡易バンドル** は、コンテナのセットで、各コンテナには全コントローラーノードに渡ってデプロイされる、同じ Pacemaker サービスが含まれます。

以下の例は、**pcs status** コマンドのバンドル設定を示しています。

```
https://gitlab.cee.redhat.com/rhci-documentation/docs-
Red_Hat_Enterprise_Virtualization/commit/bfc429f884809a197e9800312f65514c0
691b175
```

各バンドルでは、以下の情報を確認することができます。

- Pacemaker がサービスに割り当てた名前
- バンドルに関連付けられたコンテナの参照
- 異なるコントローラーで実行されているレプリカとそれらのステータスの一覧



#### 4.3.1.1. 簡易バンドルの設定

**haproxy-bundle** サービスなど、特定のバンドルサービスの詳細を確認するには、**pcs resource show** コマンドを使用します。以下に例を示します。

```
$ sudo pcs resource show haproxy-clone
Bundle: haproxy-bundle
  Docker: image=192.168.24.1:8787/rhosp12/openstack-haproxy:pcmklatest
network=host options="--user=root --log-driver=journald -e
KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=3 run-command="/bin/bash
/usr/local/bin/kolla_start"
  Storage Mapping:
    options=ro source-dir=/var/lib/kolla/config_files/haproxy.json target-
dir=/var/lib/kolla/config_files/config.json (haproxy-cfg-files)
    options=ro source-dir=/var/lib/config-data/puppet-generated/haproxy/
target-dir=/var/lib/kolla/config_files/src (haproxy-cfg-data)
    options=ro source-dir=/etc/hosts target-dir=/etc/hosts (haproxy-hosts)
    options=ro source-dir=/etc/localtime target-dir=/etc/localtime (haproxy-
localtime)
    options=ro source-dir=/etc/pki/ca-trust/extracted target-
dir=/etc/pki/ca-trust/extracted (haproxy-pki-extracted)
    options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-
dir=/etc/pki/tls/certs/ca-bundle.crt (haproxy-pki-ca-bundle-crt)
    options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-
dir=/etc/pki/tls/certs/ca-bundle.trust.crt (haproxy-pki-ca-bundle-trust-
crt)
    options=ro source-dir=/etc/pki/tls/cert.pem target-
dir=/etc/pki/tls/cert.pem (haproxy-pki-cert)
    options=rw source-dir=/dev/log target-dir=/dev/log (haproxy-dev-log)
```

**haproxy-bundle** の例では、HAProxy のリソース設定も確認できます。HAProxy は、選択したサービスへのトラフィックの負荷を分散することによって高可用性サービスを提供しますが、ここでは HAProxy を Pacemaker のバンドルサービスとして設定することによって HAProxy 自体を高可用性に保っています。

この出力例から、バンドルによって Docker コンテナがいくつかの特定のパラメーターで設定されていることがわかります。

- **image:** コンテナによって使用されるイメージ。アンダークラウドのローカルレジストリーを参照します。
- **network:** コンテナのネットワーク種別。この例では **"host"** です。
- **options:** コンテナの特定のオプション
- **replicas:** クラスター内に作成される必要のあるコンテナのコピーを数。各バンドルには、3 つのコンテナが含まれ、コントローラーノード 1 台に 1 つです。
- **run-command:** コンテナの起動に使用するシステムコマンド

Docker コンテナの仕様に加えて、バンドル設定には、ホスト上のローカルパスをコンテナにマップする **Storage Mapping** セクションも含まれています。したがって、ホストから **haproxy** の設定を確認するには、**/etc/haproxy/haproxy.cfg** ファイルの代わりに **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ファイルを開いてください。

#### 4.3.1.2. 簡易バンドルのステータスの確認

コンテナ内でコマンドを起動する **docker** コマンドを使用して、バンドルのステータスを確認できます。

```
$ sudo docker exec -it haproxy-bundle-docker-0 ps -efww | grep haproxy
root          1          0  0 Feb14 ?          00:00:00 /usr/sbin/haproxy-
systemd-wrapper -f /etc/haproxy/haproxy.cfg
haproxy       10          1  0 Feb14 ?          00:00:00 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ds
haproxy       11         10  0 Feb14 ?          00:07:47 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ds
```

この出力は、コンテナ内でプロセスが実行されていることを示しています。

ホストから直接バンドルのステータスを確認することもできます。

```
$ ps -ef | grep haproxy
root          60972      60956  0 Feb14 ?          00:00:00 /usr/sbin/haproxy-
systemd-wrapper -f /etc/haproxy/haproxy.cfg
42454         61023      60972  0 Feb14 ?          00:00:00 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ds
42454         61029      61023  0 Feb14 ?          00:07:49 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ds
heat-ad+     223079     871280  0 11:45 pts/0      00:00:00 grep --color=auto
haproxy
$ sudo ps -ef | grep [6]0956
root          60956      18734  0 Feb14 ?          00:00:00 /usr/bin/docker-
containerd-shim-current 39238c5ecb77[...]
root          60972      60956  0 Feb14 ?          00:00:00 /usr/sbin/haproxy-
systemd-wrapper -f /etc/haproxy/haproxy.cfg
$ sudo docker ps | grep haproxy-bundle
39238c5ecb77      192.168.24.1:8787/rhosp12/openstack-haproxy:pcmklatest
"/bin/bash /usr/local" 17 hours ago      Up 17 hours
haproxy-bundle-docker-0
```

検出された **haproxy** コマンドの **parent pid** 属性 (**60956**) を使用して、Docker コンテナの ID (**39238c5ecb77**) が含まれるメインの Docker プロセスを検索することができます。これは、**docker ps** コマンドの出力で表示される ID です。

任意のバンドルで同じコマンドを実行して、現在のアクティビティーレベルと、サービスによって実行されているコマンドに関する情報を確認することができます。

#### 4.3.2. 複合バンドルセットのリソース (複合バンドル)

複合バンドルセットリソースまたは **複合バンドル** は、簡易バンドルにも含まれる基本的なコンテナの設定に加えて、**リソース** 設定を指定する Pacemaker サービスです。

この追加の設定は、実行するコントローラノードによって異なる状態にすることができるサービスである **multi-State** のリソースを管理するために必要です。

以下の例には、**pcs status** コマンドで出力される複合バンドルの一覧を示しています。

```
Docker container set: rabbitmq-bundle
[192.168.24.1:8787/rhosp12/openstack-rabbitmq:pcmklatest]
```

```

    rabbitmq-bundle-0    (ocf::heartbeat:rabbitmq-cluster):      Started
overcloud-controller-0
    rabbitmq-bundle-1    (ocf::heartbeat:rabbitmq-cluster):      Started
overcloud-controller-1
    rabbitmq-bundle-2    (ocf::heartbeat:rabbitmq-cluster):      Started
overcloud-controller-2
Docker container set: galera-bundle [192.168.24.1:8787/rhosp12/openstack-
mariadb:pcmklatest]
    galera-bundle-0      (ocf::heartbeat:galera):          Master overcloud-
controller-0
    galera-bundle-1      (ocf::heartbeat:galera):          Master overcloud-
controller-1
    galera-bundle-2      (ocf::heartbeat:galera):          Master overcloud-
controller-2
Docker container set: redis-bundle [192.168.24.1:8787/rhosp12/openstack-
redis:pcmklatest]
    redis-bundle-0       (ocf::heartbeat:redis): Master overcloud-
controller-0
    redis-bundle-1       (ocf::heartbeat:redis): Slave overcloud-controller-
1
    redis-bundle-2       (ocf::heartbeat:redis): Slave overcloud-controller-
2

```

この出力では、RabbitMQ とは異なり、Galera と Redis のバンドルは multi-state リソースとしてコンテナ内で実行されていることがわかります。

**galera-bundle** リソースでは、3 つのコントローラーすべてが Galera master として実行されています。**redis-bundle** リソースでは、**overcloud-controller-0** コンテナが master として実行されている一方で、他の 2 つのコントローラーはスレーブとして実行されています。

Galera サービスは 3 つのコントローラーすべてで 1 つの制約のセットで実行されていますが、**redis** サービスはマスターとスレーブのコントローラーで異なる制約下で実行できることを意味します。

以下の例は、**pcs resource show galera-bundle** コマンドの出力を示しています。

```

[...]
Bundle: galera-bundle
  Docker: image=192.168.24.1:8787/rhosp12/openstack-mariadb:pcmklatest
masters=3 network=host options="--user=root --log-driver=journald -e
KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=3 run-command="/bin/bash
/usr/local/bin/kolla_start"
  Network: control-port=3123
  Storage Mapping:
    options=ro source-dir=/var/lib/kolla/config_files/mysql.json target-
dir=/var/lib/kolla/config_files/config.json (mysql-cfg-files)
    options=ro source-dir=/var/lib/config-data/puppet-generated/mysql/
target-dir=/var/lib/kolla/config_files/src (mysql-cfg-data)
    options=ro source-dir=/etc/hosts target-dir=/etc/hosts (mysql-hosts)
    options=ro source-dir=/etc/localtime target-dir=/etc/localtime (mysql-
localtime)
    options=rw source-dir=/var/lib/mysql target-dir=/var/lib/mysql (mysql-
lib)
    options=rw source-dir=/var/log/mariadb target-dir=/var/log/mariadb
(mysql-log-mariadb)
    options=rw source-dir=/dev/log target-dir=/dev/log (mysql-dev-log)
  Resource: galera (class=ocf provider=heartbeat type=galera)

```

```

Attributes: additional_parameters=--open-files-limit=16384
cluster_host_map=overcloud-controller-0:overcloud-controller-
0.internalapi.localdomain;overcloud-controller-1:overcloud-controller-
1.internalapi.localdomain;overcloud-controller-2:overcloud-controller-
2.internalapi.localdomain enable_creation=true
wsrep_cluster_address=gcomm://overcloud-controller-
0.internalapi.localdomain,overcloud-controller-
1.internalapi.localdomain,overcloud-controller-2.internalapi.localdomain
Meta Attrs: container-attribute-target=host master-max=3 ordered=true
Operations: demote interval=0s timeout=120 (galera-demote-interval-0s)
              monitor interval=20 timeout=30 (galera-monitor-interval-20)
              monitor interval=10 role=Master timeout=30 (galera-monitor-
interval-10)
              monitor interval=30 role=Slave timeout=30 (galera-monitor-
interval-30)
              promote interval=0s on-fail=block timeout=300s (galera-
promote-interval-0s)
              start interval=0s timeout=120 (galera-start-interval-0s)
              stop interval=0s timeout=120 (galera-stop-interval-0s)
[...]
```

この出力は、簡易バンドルとは異なり、**galera-bundle** リソースには、multi-state リソースのあらゆる側面を決定する明示的なリソース設定が含まれていることを示しています。



#### 注記

また、サービスは、同時に複数のコントローラーで実行される可能性があります、コントローラー自体は、これらのサービスに実際に到達する必要のある IP アドレスでリスンしていない場合もあります。

Galera リソースのトラブルシューティングに関する詳しい情報は「[6章 Galera の使用](#)」を参照してください。

## 4.4. PACEMAKER の FAILED ACTIONS

リソースが何らかの形で失敗した場合には、**pcs status** の出力の **Failed actions** タイトル下に記載されます。以下は、**openstack-cinder-volume** サービスが **controller-0** 上で停止した例です。

```

Failed Actions:
* openstack-cinder-volume_monitor_600000 on overcloud-controller-0 'not
running' (7): call=74, status=complete, exitreason='none',
last-rc-change='Wed Dec 14 08:33:14 2016', queued=0ms, exec=0ms
```

このような場合には、systemd サービスの **openstack-cinder-volume** を再有効化する必要があります。その他の場合には、問題を追跡/修正してからリソースをクリーンアップする必要があります。詳細は、「[コントローラー上のリソースの問題修正](#)」を参照してください。

## 4.5. コントローラーのその他の PACEMAKER 情報

**pcs status** 出力の最後のセクションでは、電源管理フェンシング (ここでは IPMI) および Pacemaker サービス自体の状態に関する情報が表示されます。

```

my-ipmilan-for-controller-0 (stonith:fence_ipmilan): Started my-ipmilan-
```

```

for-controller-0
my-ipmilan-for-controller-1 (stonith:fence_ipmilan): Started my-ipmilan-
for-controller-1
my-ipmilan-for-controller-2 (stonith:fence_ipmilan): Started my-ipmilan-
for-controller-2

PCSD Status:
overcloud-controller-0: Online
overcloud-controller-1: Online
overcloud-controller-2: Online

Daemon Status:
corosync: active/enabled
pacemaker: active/enabled openstack-cinder-volume
(systemd:openstack-cinder-volume): Started overcloud-controller-0

pcsd: active/enabled

```

**my-ipmilan-for-controller** の設定では、各ノードに指定されたフェンシングの種別 (**stonith:fence\_ipmilan**) および IPMI サービスの稼働状態が分かります。PCSD Status は、全 3 コントローラーが現在オンラインであることを示します。また、Pacemaker サービス自体には **corosync**、**pacemaker**、**pcsd** の 3 つのデーモンで構成されています。この例では、これら 3 つのサービスすべてがアクティブかつ有効化されています。

## 4.6. フェンシング用のハードウェア

コントローラーノードがヘルスチェックに失敗すると、Pacemaker 指定のコーディネーターとして機能するコントローラーは、Pacemaker **stonith** サービスを使用して、問題のあるノードをフェンシングします。Stonith は、「Shoot The Other Node In The Head」の略で、基本的に DC はクラスターからノードを除外します。

**stonith** により、フェンスデバイスが OpenStack Platform HA クラスターでどのように設定されているかを確認するには、以下のコマンドを実行します。

```

$ sudo pcs stonith show --full
Resource: my-ipmilan-for-controller-0 (class=stonith type=fence_ipmilan)
Attributes: pcmk_host_list=overcloud-controller-0 ipaddr=10.100.0.51
login=admin passwd=abc lanplus=1 cipher=3
Operations: monitor interval=60s (my-ipmilan-for-controller-0-monitor-
interval-60s)
Resource: my-ipmilan-for-controller-1 (class=stonith type=fence_ipmilan)
Attributes: pcmk_host_list=overcloud-controller-1 ipaddr=10.100.0.52
login=admin passwd=abc lanplus=1 cipher=3
Operations: monitor interval=60s (my-ipmilan-for-controller-1-monitor-
interval-60s)
Resource: my-ipmilan-for-controller-2 (class=stonith type=fence_ipmilan)
Attributes: pcmk_host_list=overcloud-controller-2 ipaddr=10.100.0.53
login=admin passwd=abc lanplus=1 cipher=3
Operations: monitor interval=60s (my-ipmilan-for-controller-2-monitor-
interval-60s)

```

**show --full** の一覧では、フェンシングに関連するコントローラーノード 3 つの詳細が表示されます。フェンスデバイスは、IPMI 電源管理 (**fence\_ipmilan**) を使用して、必要に応じてマシンの電源のオン/オフを切り替えます。各ノードの IPMI インターフェースに関する情報には、IPMI インターフェース

の IP アドレス (**10.100.0.51**)、ログインするユーザー名 (**admin**)、使用するパスワード (**abc**) が含まれます。各ホストが監視される間隔も (60 秒) 確認することができます。

Pacemaker を使用したフェンシングに関する詳しい情報は、『**Red Hat Enterprise Linux 7 High Availability Add-On の管理**』の「[フェンシングの設定](#)」を参照してください。

## 第5章 HAPROXY の使用

HAProxy は、トラフィックの負荷を複数のコントローラーに分散することによって、OpenStack に高可用性機能を提供します。**haproxy** パッケージには、ロギング機能やサンプルの設定以外に、**systemd** サービスから起動される **haproxy** デーモンが含まれています。前述したように、Pacemaker は HAProxy サービス自体を、**haproxy-bundle** と呼ばれる高可用性サービスとして管理します。



### 注記

HAProxy の設定を検証する方法については、「[haproxy.cfg が OpenStack のサービスをロードバランシングできるように正しく設定されているかどうか、確認する方法はありますか?](#)」の KCS ソリューションを参照してください。

Red Hat OpenStack Platform では、director により複数の OpenStack サービスが haproxy サービスを有効活用できるように設定されます。HAProxy は各オーバークラウドノード上の専用のコンテナ内で実行されるため、director はそれらの OpenStack サービスを **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg** ファイルで構成することによって設定します。

そのファイル内の各サービスで、以下のプロパティを確認できます。

- **listen**: 要求をリッスンするサービス名
- **bind**: サービスがリッスンする IP アドレスおよび TCP ポート番号
- **server**: サービスを提供する各サーバー名、サーバーの IP アドレス、リッスンするポート、その他の情報

director での Red Hat OpenStack Platform のインストール時に作成される **haproxy.cfg** ファイルにより、HAProxy が管理する 19 の異なるサービスが特定されます。**haproxy.cfg** ファイルでの **cinder listen** サービスの設定方法の例を以下に示します。

```
listen cinder
    bind 172.16.0.10:8776
    bind 192.168.1.150:8776
    mode http
    http-request set-header X-Forwarded-Proto https if { ssl_fc }
    http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
    option httpchk
    server overcloud-controller-0 172.16.0.13:8777 check fall 5 inter 2000
rise 2
    server overcloud-controller-1 172.16.0.14:8777 check fall 5 inter 2000
rise 2
    server overcloud-controller-2 172.16.0.15:8777 check fall 5 inter 2000
rise 2
```

上記の Cinder サービスの HAProxy 設定例では、Cinder サービスが提供されている IP アドレスとポートを特定できます (ポート 8777 は 172.16.0.10 と 192.168.1.150)。

172.16.0.10 アドレスは、オーバークラウド内で使用するための内部 API ネットワーク (VLAN201) 上の仮想 IP アドレスです。192.168.1.150 は、オーバークラウドの外部からの API ネットワークへのアクセスを提供する外部ネットワーク (VLAN100) 上の仮想 IP アドレスです。

HAProxy は、これらの 2 つのアドレスに対する要求を **overcloud-controller-0** (172.16.0.13:8777)、**overcloud-controller-1** (172.16.0.14:8777)、**overcloud-controller-2** (172.16.0.15:8777) のいずれかに転送することができます。

これらのサーバーに設定されたオプションでは、ヘルスチェック (**check**) が有効になり、ヘルスチェックに 5 回失敗すると (**fall 5**)、サービスは停止されていると見なされます。ヘルスチェックの実行する間隔は、**inter 2000** (2000 ミリ秒または 2 秒) に設定されます。また、ヘルスチェックに 2 回成功すると (**rise 2**)、サーバーは稼働していると見なされます。

コントローラーノードで HAProxy が管理するサービスの一覧を以下に示します。

**表5.1 HAProxy が管理するサービス**

|                |                 |            |                |
|----------------|-----------------|------------|----------------|
| aodh           | cinder          | glance_api | gnocchi        |
| haproxy.stats  | heat_api        | heat_cfn   | horizon        |
| keystone_admin | keystone_public | mysql      | neutron        |
| nova_metadata  | nova_novncproxy | nova_osapi | nova_placement |

## 5.1. HAPROXY STATS

director により、HA デプロイメントではすべて、**HAProxy Stats** もデフォルトで有効になります。この機能により、データ転送、接続、サーバーの状態などについての詳細情報を HAProxy Stats のページで確認することができます。

また、director は、HAProxy Stats ページにアクセスするための IP:Port アドレスも設定します。このアドレスを確認するには、HAProxy がインストールされている任意のノードで **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg** ファイルを開くと、**listen haproxy.stats** セクションにこの情報が記載されています。以下に例を示します。

```
listen haproxy.stats
    bind 10.200.0.6:1993
    mode http
    stats enable
    stats uri /
    stats auth admin:<haproxy-stats-password>
```

この場合は、ブラウザで **10.200.0.6:1993** にナビゲートして、**stats auth** の行から認証情報を入力すると、HAProxy Stats のページが表示されます。

## 5.2. 参考資料

HAProxy に関する詳しい情報は、「[HAProxy の設定](#)」(『[ロードバランサーの管理](#)』)を参照してください。

**haproxy.cfg** ファイルで使用可能な設定に関する詳しい情報は、**haproxy** パッケージがインストールされている任意のシステム (例: コントローラーノードなど) の **/usr/share/doc/haproxy-  
\_VERSION/configuration.txt\_** のドキュメントを参照してください。



## 第6章 GALERA の使用

高可用性のデプロイメントでは、Red Hat OpenStack Platform は [MariaDB Galera Cluster](#) を使用してデータベースのレプリケーションを管理します。「[Pacemaker で設定された OpenStack サービス](#)」に記載のとおり、Pacemaker は、マスター/スレーブのステータスを管理する **バンドルセット** リソースを使用して Galera サービスを実行します。

**pcs status** コマンドを使用して、**galera-bundle** サービスが実行されているかどうかと、実行先のコントローラーを確認することができます。

```
Docker container set: galera-bundle [192.168.24.1:8787/rhosp12/openstack-
mariadb:pcmklatest]
  galera-bundle-0      (ocf::heartbeat:galera):      Master overcloud-
controller-0
  galera-bundle-1      (ocf::heartbeat:galera):      Master overcloud-
controller-1
  galera-bundle-2      (ocf::heartbeat:galera):      Master overcloud-
controller-2
```

### 6.1. ホスト名の解決

MariaDB Galera Cluster のトラブルシューティングを行う際には、ホスト名の解決を確認することができます。デフォルトでは、director は Galera リソースを IP アドレスではなく **hostname** にバインドします。<sup>[1]</sup>そのため、ホスト名の解決を妨げている問題 (例: 設定の誤りや DNS のエラーなど) があると、Pacemaker が Galera リソースを適切に管理できなくなる可能性があります。

ホスト名の解決の問題がないことを確認した後には、クラスター自体の整合性をチェックしてください。そのためには、各コントローラーノードのデータベースで、Write Set Replication のステータスを確認します。

MySQL にアクセスするには、オーバークラウドのデプロイメント中に director によって設定されたパスワードを使用します。MySQL の root パスワードを取得するには、**hiera** コマンドを使用します。

```
$ sudo hiera -c /etc/puppet/hiera.yaml "mysql::server::root_password"
*<MYSQL-HIERA-PASSWORD>*
```

Write Set Replication の情報は、各ノードの MariaDB データベースに保管されます。関連する変数はそれぞれ **wsrep\_** のプレフィックスを使用します。そのため、この情報はデータベースクライアントで直接問い合わせることができます。

```
$ sudo mysql -B --password="<MYSQL-HIERA-PASSWORD>" -e "SHOW GLOBAL STATUS
LIKE 'wsrep_%';"
+-----+
| Variable_name      | Value |
+-----+
| wsrep_protocol_version | 5      |
| wsrep_last_committed | 202    |
| ...                | ...    |
| wsrep_thread_count  | 2      |
+-----+
```

MariaDB Galera Cluster の正常性と整合性を検証するには、まず最初にクラスターが正しいノード数を報告するかどうかを確認してから、各ノードで以下の点をチェックします。

- 正しいクラスターに属していること
- クラスターへの書き込みが可能であること
- クラスターからクエリーの受信と書き込みが可能であること
- クラスター内の他のノードに接続されていること
- ローカルのデータベースで Write Set をテーブルにレプリケーションしていること

## 6.2. データベースクラスターの整合性

MariaDB Galera Cluster の問題を調査する際には、クラスター自体の整合性を確認することができます。クラスターの整合性を検証するには、各コントローラーノード上の特定の **wsrep\_** データベース変数を確認する必要があります。データベースの変数を確認するには、次のコマンドを実行します。

```
$ sudo mysql -B --password="<MYSQL-HIERA-PASSWORD>" -e "SHOW GLOBAL STATUS LIKE 'VARIABLE';"
```

**VARIABLE** は確認する **wsrep\_** データベース変数に置き換えてください。たとえば、ノードの cluster state UUID を確認するには、以下のコマンドを実行します。

```
$ sudo mysql -B --password="<MYSQL-HIERA-PASSWORD>" -e "SHOW GLOBAL STATUS LIKE 'wsrep_cluster_state_uuid';"
+-----+
+
+ | Variable_name          | Value                                     |
+-----+-----+
+ | wsrep_cluster_state_uuid | e2c9a15e-5485-11e0-0800-6bbb637e7211 |
+-----+-----+
+
```

以下の表には、クラスターの整合性に関連するさまざまな **wsrep\_** データベース変数をまとめています。

表6.1 クラスターの整合性を確認するためのデータベース変数

| 変数                              | 概要             | 説明   |
|---------------------------------|----------------|--|
| <b>wsrep_cluster_state_uuid</b> | クラスターの状態の UUID | ノードが属するクラスターの ID。全ノードに全く同じ ID が割り当てられている必要があります。異なる ID が割り当てられたノードは、クラスターに接続できません。 |
| <b>wsrep_cluster_size</b>       | クラスター内のノード数    | これは、任意のノード 1 台で確認することができます。値が実際のノード数を下回る場合には、いずれかのノードでエラーが発生したか、接続を失ったこととなります。     |

| 変数                                 | 概要                    | 説明   |
|------------------------------------|-----------------------|--|
| <code>wsrep_cluster_conf_id</code> | クラスターの変更回数            | <p>クラスターが複数のコンポーネント (パーティション) に分割されているかどうかを確認します。これは、ネットワークのエラーが原因となっている場合が多いです。全ノードの値が全く同じである必要があります。</p> <p>異なる <code>wsrep_cluster_conf_id</code> を報告するノードがある場合には、<code>wsrep_cluster_status</code> の値をチェックして、クラスター (<b>Primary</b>) への書き込みができるかどうかを確認してください。</p> |
| <code>wsrep_cluster_status</code>  | Primary コンポーネントのステータス | <p>ノードがまだクラスターへの書き込みをできるかどうかを確認します。可能な場合には、<code>wsrep_cluster_status</code> は <b>Primary</b> のはずです。それ以外の値の場合には、ノードが稼働していないパーティションの一部であることを示します。</p>   |

### 6.3. データベースクラスターノード

Galera クラスターの問題を特定のノードに切り分けすることができる場合には、その他の `wsrep_` データベース変数が具体的な問題を解く手がかりとなる可能性があります。これらの変数は、「[データベースクラスターの整合性](#)」に記載のクラスターのチェックと同様の方法で確認することができます。

```
$ sudo mysql -B --password="<MYSQL-HIERA-PASSWORD>" -e "SHOW GLOBAL STATUS LIKE 'VARIABLE';"
```

**VARIABLE** は以下の値のいずれかに置き換えます。

表6.2 ノードの整合性を確認するためのデータベース変数

| 変数                       | 概要               | 説明   |
|--------------------------|------------------|--|
| <code>wsrep_ready</code> | ノードがクエリーを受け入れる機能 | <p>ノードがクラスターから write set を受け入れることができるかどうかを示します。受け入れることができる場合には、<code>wsrep_ready</code> は <b>ON</b> のはずです。</p> |

| 変数                        | 概要            | 説明  |
|---------------------------|---------------|---|
| wsrep_connected           | ノードのネットワーク接続性 | そのノードが他のノードに接続されているかどうかを示します。接続されている場合には、 <b>wsrep_connected</b> は <b>ON</b> のはずです。   |
| wsrep_local_state_comment | ノードの状態        | ノードの状態の概要を表示します。ノードがクラスターにまだ書き込み可能な場合には ( <b>wsrep_cluster_status</b> が <b>Primary</b> の場合には「 <a href="#">データベースクラスタの整合性</a> 」を参照)、 <b>wsrep_local_state_comment</b> の一般的な値は <b>Joining</b> 、 <b>Waiting on SST</b> 、 <b>Joined</b> 、 <b>Synced</b> 、 <b>Donor</b> のいずれかです。<br><br>ノードが稼働していないコンポーネントの一部の場合には、 <b>wsrep_local_state_comment</b> は <b>Initialized</b> に設定されます。 |

### 注記

**wsrep\_connected** が **ON** の場合には、そのノードが一部のノードのみに接続されている可能性があることも意味します。たとえば、クラスターのパーティションの場合には、そのノードは、クラスターに書き込みができないコンポーネントの一部となっている可能性があります。詳しくは、「[データベースクラスタの整合性](#)」を参照してください。

**wsrep\_connected** が **OFF** の場合は、ノードはどのクラスターコンポーネントにも接続されていません。

## 6.4. データベースレプリケーションのパフォーマンス

クラスターおよびそのクラスター内の個々のノードが正常な状態で安定している場合には、レプリケーションのスループットを確認して、パフォーマンスのベンチマークを行うことができます。この作業は、「[データベースクラスタノード](#)」および「[データベースクラスタの整合性](#)」と同様に **wsrep\_** データベースの変数をチェックすることによって行います。

```
$ sudo mysql -B --password="<MYSQL-HIERA-PASSWORD>" -e "SHOW STATUS LIKE 'VARIABLE';"
```

**VARIABLE** は以下の値のいずれかに置き換えます。

表6.3 クラスターのパフォーマンス (レプリケーションのスループット) を確認するためのデータベース変数

| 変数  | 概要  |
|---|---|
| <code>wsrep_local_recv_queue_avg</code>   | 最後にクエリーされた後のローカル受信キューの平均サイズ                                   |
| <code>wsrep_local_send_queue_avg</code>   | その変数が最後にクエリーされた後の送信キューの平均の長さ                                  |
| <code>wsrep_local_recv_queue_min</code> および <code>wsrep_local_recv_queue_max</code> | いずれかの変数が最後にクエリーされた後のローカル受信キューの最小サイズと最大サイズ                     |
| <code>wsrep_flow_control_paused</code>  | その変数が最後にクエリーされた後に フロー制御 が原因でノードが一時停止された時間の割合                  |
| <code>wsrep_cert_deps_distance</code>   | 並行して適用可能なシーケンス番号 ( <b>seqno</b> ) の最小値と最大値の幅の平均 (例: 潜在的な並列化度) |

それらの変数のいずれかをクエリーするときには毎回、**FLUSH STATUS** コマンドを実行すると値がリセットされます。クラスターのレプリケーションのベンチマークを行うには、それらの値を複数回クエリーして、その変化を確認する必要があります。この変化は、**フロー制御** がクラスターのパフォーマンスにどの程度影響を及ぼしているかを判断するのに役立てることができます。

フロー制御とは、クラスターがレプリケーションを制御するのに使用するメカニズムです。ローカル受信 Write Set キューが一定の閾値を超えると、ノードがそのキューに追い付くために、フロー制御がレプリケーションを一時停止します。詳しくは、[Galera Cluster](#) のサイトで、「[Flow Control](#)」のセクションを参照してください。

異なる値およびベンチマークを最適化するには、以下の変数をチェックします。

#### `wsrep_local_recv_queue_avg > 0.0`

ノードが 受信に対応できる速度で Write Set を適用することはできないため、**レプリケーションのスロットル**がトリガーされます。このベンチマークの詳しい情報は、`wsrep_local_recv_queue_min` と `wsrep_local_recv_queue_max` を確認してください。

#### `wsrep_local_send_queue_avg > 0.0`

`wsrep_local_send_queue_avg` の値が高くなると、レプリケーションのスロットルとネットワークスループットの問題が発生する可能性も高くなります。これは特に `wsrep_local_recv_queue_avg` の値が高くなると、その傾向が強くなります。

#### `wsrep_flow_control_paused > 0.0`

フロー制御によりノードが一時停止されています。ノードが一時停止されている時間を確認するには、`wsrep_flow_control_paused` の値をクエリーの間隔の秒数で乗算してください。たとえば、最後のチェックから 1 分後に `wsrep_flow_control_paused = 0.50` となった場合には、ノードのレプリケーションは 30 秒停止されたことになります。また、`wsrep_flow_control_paused = 1.0` の場合には、最後のクエリーからずっと停止していたことになります。

理想的には、`wsrep_flow_control_paused` は可能な限り **0.0** に近い値であるべきです。

#### `wsrep_cert_deps_distance`

スロットルおよび一時停止の場合には、**wsrep\_cert\_deps\_distance** の変数をチェックして、(平均で) 適用可能な write set の数を確認することができます。その後、**wsrep\_slave\_threads** をチェックして、同時に適用された write set の実際の数を確認します。

### **wsrep\_slave\_threads**

**wsrep\_slave\_threads** の設定をより高くすると、スロットリングと一時停止を緩和するのに役立ちます。たとえば、**wsrep\_cert\_deps\_distance** の値が **20** の場合は、**wsrep\_slave\_threads** を 2 倍にして **2** から **4** にすると、ノードが適用できる write set の量も 2 倍になります。ただし、**wsrep\_slave\_threads** は、ノード内の CPU コア数を超えないように設定する必要があります。問題のあるノードで、**wsrep\_slave\_threads** がすでに最適に設定されている場合には、接続の問題の可能性を調査するためにそのノードをクラスターから除外することを検討してください。

---

[1] このメソッドは、IPv6 を使用するオーバークラウドで Galera が正常に起動できるようにするために実装されました (詳しくは、[BZ#1298671](#) を参照してください)。

## 第7章 HA コントローラーリソースの調査と修正

**pcs constraint show** コマンドは、サービスの起動方法に対する制約を表示します。このコマンドの出力には、各リソースの場所、リソースの起動順序、別のリソースと一緒に起動する必要があるのかなど、制約が表示されます。問題がある場合には、これらの問題を修正してから、リソースをクリーンアップします。

以下の例には、コントローラーで **pcs constraint show** を実行した場合の出力を示しており、途中省略しています。

```
$ sudo pcs constraint show
Location Constraints:
  Resource: galera-bundle
    Constraint: location-galera-bundle (resource-discovery=exclusive)
    Rule: score=0
    Expression: galera-role eq true
  [...]
  Resource: ip-192.168.24.15
    Constraint: location-ip-192.168.24.15 (resource-discovery=exclusive)
    Rule: score=0
    Expression: haproxy-role eq true
  [...]
  Resource: my-ipmilan-for-controller-0
    Disabled on: overcloud-controller-0 (score:-INFINITY)
  Resource: my-ipmilan-for-controller-1
    Disabled on: overcloud-controller-1 (score:-INFINITY)
  Resource: my-ipmilan-for-controller-2
    Disabled on: overcloud-controller-2 (score:-INFINITY)
Ordering Constraints:
  start ip-172.16.0.10 then start haproxy-bundle (kind:Optional)
  start ip-10.200.0.6 then start haproxy-bundle (kind:Optional)
  start ip-172.19.0.10 then start haproxy-bundle (kind:Optional)
  start ip-192.168.1.150 then start haproxy-bundle (kind:Optional)
  start ip-172.16.0.11 then start haproxy-bundle (kind:Optional)
  start ip-172.18.0.10 then start haproxy-bundle (kind:Optional)
Colocation Constraints:
  ip-172.16.0.10 with haproxy-bundle (score:INFINITY)
  ip-172.18.0.10 with haproxy-bundle (score:INFINITY)
  ip-10.200.0.6 with haproxy-bundle (score:INFINITY)
  ip-172.19.0.10 with haproxy-bundle (score:INFINITY)
  ip-172.16.0.11 with haproxy-bundle (score:INFINITY)
  ip-192.168.1.150 with haproxy-bundle (score:INFINITY)
```

この出力には、以下の3つの主要な制約のタイプが表示されています。

### Location Constraints

このセクションには、リソースを割り当てる場所に関する制約が示されています。最初の制約は、**galera-role** 属性が **true** に設定されたノードで実行される **galera-bundle** リソースを設定するルールを定義します。**pcs property show** コマンドを使用して、ノードの属性を確認することができます。

```
$ sudo pcs property show
Cluster Properties:
  cluster-infrastructure: corosync
  cluster-name: tripleo_cluster
```

```

dc-version: 1.1.16-12.el7_4.7-94ff4df
have-watchdog: false
redis_REPL_INFO: overcloud-controller-0
stonith-enabled: false
Node Attributes:
  overcloud-controller-0: cinder-volume-role=true galera-role=true
  haproxy-role=true rabbitmq-role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-0
  overcloud-controller-1: cinder-volume-role=true galera-role=true
  haproxy-role=true rabbitmq-role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-1
  overcloud-controller-2: cinder-volume-role=true galera-role=true
  haproxy-role=true rabbitmq-role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-2

```

この出力から、全コントローラーの **galera-role** 属性が **true** であることを確認できます。これは、**galera-bundle** リソースがこれらのノードでのみ実行されることを意味します。これと同じ概念が、場所に関する他の制約に関連付けられたその他の属性に適用されます。

場所に関する 2 番目の制約は、リソース `ip-192.168.24.15` に関連し、**haproxy-role** 属性が **true** に設定されたノードでのみ IP リソースを実行するように指定しています。これは、クラスターが IP アドレスを **haproxy** サービスに関連付けることを意味し、サービスを到達可能にするために必要です。

場所に関する 3 番目の制約には、各コントローラーで **ipmilan** リソースが無効化されていることを意味します。

## Ordering Constraints

このセクションでは、仮想 IP アドレスリソース (`IPAddr2`) が `HAProxy` より前に起動するように強制する制約を示しています。Ordering Constraints は、IP アドレスリソースと `HAProxy` にのみ適用されます。Compute などの各サービスは、独立したサービス (例: Galera) の中断をサポートすることができると想定されているので、その他のリソースはすべて `systemd` によって管理されます。

## Co-location Constraints

このセクションでは、どのリソースを併置する必要があるかが表示されています。すべての仮想 IP アドレスが **haproxy-bundle** リソースにリンクされています。

## 7.1. コントローラー上のリソースの問題修正

クラスターによって管理されているリソースに関する失敗したアクションは、**pcs status** コマンドで表示されます。多くの異なる問題が発生する可能性があります。一般的には、以下の方法で問題に対処することができます。

### コントローラーの問題

コントローラーのヘルスチェックでエラーが発生した場合には、そのコントローラーにログインしてサービスが問題なく起動できるかどうかを確認してください。サービスの起動で問題がある場合には、コントローラー間での通信の問題がある可能性があることになります。コントローラー間の通信問題のその他の兆候には、以下が含まれます。

- 1 台のコントローラーが他のコントローラーよりも過度にフェンシングされる
- 特定のコントローラーから、不審な大量のサービスでエラーが発生している

### 個別のリソースの問題



コントローラーからのサービスが基本的には機能しているが、個別のリソースでエラーが発生している場合には、**pcs status** のメッセージで問題を特定できるか確認します。さらに情報が必要な場合には、リソースの問題があるコントローラーにログインして、以下のステップのいくつかを試してください。

IP とコアバンドルリソース (Galera、Rabbit、Redis) 以外で、クラスターによって管理されている唯一の A/P リソースは **openstack-cinder-volume** です。このリソースに、関連付けられた失敗したアクションがある場合の適切な対処方法は **systemctl** 側からチェックすることです。リソースが失敗しているノード (例: **overcloud-controller-0**) を特定した後は、そのリソースのステータスをチェックすることができます。

```
[heat-admin@overcloud-controller-0 ~]$ sudo systemctl status openstack-cinder-volume
● openstack-cinder-volume.service - Cluster Controlled openstack-cinder-volume
   Loaded: loaded (/usr/lib/systemd/system/openstack-cinder-volume.service; disabled; vendor preset: disabled)
   Drop-In: /run/systemd/system/openstack-cinder-volume.service.d
            └─50-pacemaker.conf
   Active: active (running) since Tue 2016-11-22 09:25:53 UTC; 2 weeks 6 days ago
   Main PID: 383912 (cinder-volume)
   CGroup: /system.slice/openstack-cinder-volume.service
           └─383912 /usr/bin/python2 /usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/volume.log
           └─383985 /usr/bin/python2 /usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/volume.log
```

```
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:55.798 383912 WARNING oslo_config.cfg [req-8f32db96-7ca2-4fc5-82ab-271993b28174 - - - -...e future.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:55.799 383912 WARNING oslo_config.cfg [req-8f32db96-7ca2-4fc5-82ab-271993b28174 - - - -...e future.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:55.926 383985 INFO cinder.coordination [-] Coordination backend started successfully.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:55.926 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - - ...r (1.2.0)
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:56.047 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:56.048 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:56.048 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22 09:25:56.063 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - - ...essfully.
```

```
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-  
volume[383912]: 2016-11-22 09:25:56.111 383985 INFO  
cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - - ...r  
(1.2.0)  
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-  
volume[383912]: 2016-11-22 09:25:56.146 383985 INFO  
cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-3d2bfc98ecb5 - -  
...essfully.  
Hint: Some lines were ellipsized, use -l to show in full.
```

エラーが発生したリソースを修正した後には、**pcs resource cleanup** コマンドを実行して、リソースの状態と失敗回数をリセットすることができます。その次に、**httpd-cloneopenstack-cinder-volume** リソースの問題を特定および修正してから、以下のコマンドを実行します。

```
$ sudo pcs resource cleanup openstack-cinder-volume  
Resource: openstack-cinder-volume successfully cleaned up
```

## 第8章 HA CEPH ノードの調査

Ceph ストレージをデプロイする場合には、Red Hat OpenStack Platform では **ceph-mon** を Ceph クラスター用のモニターデーモンとして使用します。director はこのデーモンを全コントローラーノードにデプロイします。

Ceph のモニタリングサービスが稼働中であることを確認するには、以下のコマンドを実行してください。

```
$ sudo service ceph status
=== mon.overcloud-controller-0 ===
mon.overcloud-controller-0: running {"version":"0.94.1"}
```

コントローラーおよび Ceph ノード上の **/etc/ceph/ceph.conf** ファイルを参照すると、Ceph がどのように設定されているかを確認することができます。以下に例を示します。

```
[global]
osd_pool_default_pgp_num = 128
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-
1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```

この例では、3 台のコントローラーノード (**overcloud-controller-0**、**-1**、**-2**) がすべて Ceph クラスター (**mon\_initial\_members**) を監視するように設定されています。172.19.0.11/24 ネットワーク (VLAN 203) はストレージ管理ネットワークとして使用され、コントローラーと Ceph Storage ノード間の通信パスを提供します。

これら 3 つの Ceph Storage ノードは、別のネットワーク上にあります。上記の例からもわかるように、これら 3 つのノードの IP アドレスは、ストレージネットワーク (VLAN 202) 上にあり、172.18.0.15、172.18.0.16、172.18.0.17 として定義されています。

Ceph ノードの現在のステータスを確認するには、ノードにログインして以下のコマンドを実行します。

```
# ceph -s
  cluster 8c835acc-6838-11e5-bb96-2cc260178a92
    health HEALTH_OK
      monmap e1: 3 mons at {overcloud-controller-
0=172.18.0.17:6789/0,overcloud-controller-1=172.18.0.15:6789/0,overcloud-
controller-2=172.18.0.16:6789/0}
        election epoch 152, quorum 0,1,2 overcloud-controller-
1,overcloud-controller-2,overcloud-controller-0
      osdmap e543: 6 osds: 6 up, 6 in
```

```
pgmap v1736: 256 pgs, 4 pools, 0 bytes data, 0 objects
             267 MB used, 119 GB / 119 GB avail
             256 active+clean
```

**ceph -s** の出力から、Ceph クラスターの正常性に問題がないこと (**HEALTH\_OK**) が分かります。Ceph モニターサービスは 3 つあります (3 台の **overcloud-controller** ノードで実行されています)。また、それぞれがリッスンする IP アドレスとポートも表示されています。

Red Hat Ceph に関する詳しい情報は、[Red Hat Ceph の製品ページ](#)を参照してください。

## 付録A RED HAT OPENSTACK PLATFORM 13 HA 環境の構築

『[Deploying an Overcloud with Containerized Red Hat Ceph](#)』ガイドには、本ガイドで説明しているタイプの高可用性 OpenStack 環境のデプロイの手順が記載されています。また、『[director のインストールと使用方法](#)』ガイドも全プロセスにわたって参考のために使用しました。

### A.1. ハードウェアの仕様

以下の表には、本ガイドで検証するデプロイメントに使う仕様をまとめています。より良い結果を出すには、お使いのテストデプロイメントで CPU、メモリー、ストレージ、NIC を増やしてください。

表A.1 物理コンピューター

| コンピューターの台数 | 割り当てられた用途             | CPU | メモリー    | ディスク空き容量 | 電源管理 | NIC  |
|------------|-----------------------|-----|---------|----------|------|--|
| 1          | director ノード          | 4   | 6144 MB | 40 GB    | IPMI | 2 (外部 x 1、プロビジョニング x 1) + 1 IPMI               |
| 3          | コントローラーノード            | 4   | 6144 MB | 40 GB    | IPMI | 3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI |
| 3          | Ceph Storage ノード      | 4   | 6144 MB | 40 GB    | IPMI | 3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI |
| 2          | コンピューターノード (必要に応じて追加) | 4   | 6144 MB | 40 GB    | IPMI | 3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI |

以下の一覧では、director 以外に割り当てられているノードに関連付けられた一般的な機能と接続について説明します。

#### コントローラーノード

ストレージ以外の OpenStack サービスの多くは、コントローラーノード上で稼働します。全サービスは、この 3 つのノードで複製されます (Active/Active や Active/Passive)。3 つのノードには、信頼性の高い HA が必要です。

### Ceph Storage ノード

ストレージサービスは Ceph Storage ノードで稼働して、Ceph Storage 領域プールをコンピュートノードに提供します。この場合も、3 つのノードには、HA を指定する必要があります。

### コンピュートノード

仮想マシンは、実際にはコンピュートノードで稼働します。コンピュートノードのシャットダウンやノード間の仮想マシンの移行機能など、キャパシティー要件を満たすのに必要とされる数のコンピュートノードを指定することができます。仮想マシンがストレージにアクセスできるようにするには、コンピュートノードをストレージネットワークに接続する必要があります。また、仮想マシンが他のコンピュートノード上の仮想マシンや交換ネットワークにアクセスしてサービスを利用できるようにするには、コンピュートノードをテナントネットワークに接続する必要があります。

表A.2 物理および仮想ネットワーク

| 物理 NIC        | ネットワークの理由                 | VLAN     | 用途  |
|---------------|---------------------------|----------|---|
| eth0          | プロビジョニングネットワーク (アンダークラウド) | N/A      | director からの全ノードの管理 (アンダークラウド)                                |
| eth1 および eth2 | コントローラー/外部 (オーバークラウド)     | N/A      | VLAN を使用したボンディング NIC  |
|               | 外部ネットワーク                  | VLAN 100 | 外部環境からテナントネットワーク、内部 API、OpenStack Horizon Dashboard へのアクセスの許可 |
|               | 内部 API                    | VLAN 201 | コンピュートノードとコントローラーノード間の内部 API へのアクセス提供                         |
|               | ストレージアクセス                 | VLAN 202 | コンピュートノードと下層のストレージメディアとの接続                                    |
|               | ストレージ管理                   | VLAN 203 | ストレージメディアの管理  |
|               | テナントネットワーク                | VLAN 204 | OpenStack に対するテナントネットワークの提供                                   |

以下のハードウェアも必要です。

### プロビジョニングネットワーク用のスイッチ

このスイッチは、director システム (アンダークラウド) を Red Hat OpenStack Platform 環境の全コ

ンピューター (オーバークラウド) に接続する必要があります。このスイッチに接続されている、各オーバークラウドノードの NIC は、director から PXE ブートできなければなりません。また、スイッチの portfast が有効に設定されていることを確認してください。

### コントローラー/外部ネットワーク用のスイッチ

このスイッチは、図 1 に示した VLAN 用に VLAN タグ付けをするように設定する必要があります。外部ネットワークには、VLAN 100 のトラフィックのみを許可すべきです。

### フェンシング用のハードウェア

この構成では、Pacemaker とともに使用するように定義されたハードウェアがサポートされています。サポートされているフェンシングデバイスは、**stonith** Pacemaker ツールを使用して特定することができます。詳しい情報は、「[Fencing the Controller Nodes](#)」を参照してください。

## A.2. アンダークラウドの設定ファイル

以下の項には、本ガイドで使用しているテスト設定の関連する設定ファイルを記載します。IP アドレスの範囲を変更する場合には、[図1.1 「director を使用してデプロイした OpenStack HA 環境」](#)のような図を作成して、変更後のアドレス設定を記録することを検討してください。

### instackenv.json

```
{
  "nodes": [
    {
      "pm_password": "testpass",
      "memory": "6144",
      "pm_addr": "10.100.0.11",
      "mac": [
        "2c:c2:60:3b:b3:94"
      ],
      "pm_type": "pxe_ipmitool",
      "disk": "40",
      "arch": "x86_64",
      "cpu": "1",
      "pm_user": "admin"
    },
    {
      "pm_password": "testpass",
      "memory": "6144",
      "pm_addr": "10.100.0.12",
      "mac": [
        "2c:c2:60:51:b7:fb"
      ],
      "pm_type": "pxe_ipmitool",
      "disk": "40",
      "arch": "x86_64",
      "cpu": "1",
      "pm_user": "admin"
    },
    {
      "pm_password": "testpass",
      "memory": "6144",
      "pm_addr": "10.100.0.13",
      "mac": [
        "2c:c2:60:76:ce:a5"
      ],
    },
  ],
}
```

```
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.51",
    "mac": [
      "2c:c2:60:08:b1:e2"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.52",
    "mac": [
      "2c:c2:60:20:a1:9e"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.53",
    "mac": [
      "2c:c2:60:58:10:33"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.101",
    "mac": [
      "2c:c2:60:31:a9:55"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "2",
```



```

        "pm_user": "admin"
    },
    {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.102",
        "mac": [
            "2c:c2:60:0d:e7:d1"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "2",
        "pm_user": "admin"
    }
],
"overcloud": {"password":
"7adbbbeedc5b7a07ba1917e1b3b228334f9a2d4e",
"endpoint": "http://192.168.1.150:5000/v2.0/"
}
}

```

### undercloud.conf

```

[DEFAULT]
image_path = /home/stack/images
local_ip = 10.200.0.1/24
undercloud_public_vip = 10.200.0.2
undercloud_admin_vip = 10.200.0.3
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
local_interface = eth0
masquerade_network = 10.200.0.0/24
dhcp_start = 10.200.0.5
dhcp_end = 10.200.0.24
network_cidr = 10.200.0.0/24
network_gateway = 10.200.0.1
#discovery_interface = br-ctlplane
discovery_iprange = 10.200.0.150,10.200.0.200
discovery_runbench = 1
undercloud_admin_password = testpass
...

```

### network-environment.yaml

```

resource_registry:
  OS::Triple0::BlockStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/cinder-storage.yaml
  OS::Triple0::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::Triple0::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
  OS::Triple0::ObjectStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/swift-storage.yaml
  OS::Triple0::CephStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/ceph-storage.yaml

```

```

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ExternalNetCidr: 192.168.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end':
'172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end':
'172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end':
'172.19.0.200'}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{'start': '192.168.1.150', 'end':
'192.168.1.199'}]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ExternalNetworkVlanID: 100
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.1.1
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: ""
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=active-backup lacp=off other_config:bond-miimon-
interval=100"

```

### A.3. オーバークラウドの設定ファイル

以下の設定ファイルは、本ガイドで使用しているデプロイメントの実際にオーバークラウドの設定を反映しています。

#### **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg (Controller Nodes)**

このファイルは、HAProxy が管理するサービスを特定します。これには、HAProxy によってモニタリングされるサービスを定義する設定が記載されています。このファイルは、コントローラーノード上に存在し、全コントローラーノードで同じ内容となります。

```

# This file is managed by Puppet
global
  daemon
  group haproxy
  log /dev/log local0
  maxconn 20480
  pidfile /var/run/haproxy.pid
  ssl-default-bind-ciphers
!SSLv2:kEECDH:kRSA:kEDH:kPSK:+3DES:!aNULL:!eNULL:!MD5:!EXP:!RC4:!SEED:!IDE
A:!DES
  ssl-default-bind-options no-sslsv3
  stats socket /var/lib/haproxy/stats mode 600 level user
  stats timeout 2m
  user haproxy

```

```

defaults
  log global
  maxconn 4096
  mode tcp
  retries 3
  timeout http-request 10s
  timeout queue 2m
  timeout connect 10s
  timeout client 2m
  timeout server 2m
  timeout check 10s

listen aodh
  bind 192.168.1.150:8042 transparent
  bind 172.16.0.10:8042 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8042
check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8042
check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8042
check fall 5 inter 2000 rise 2

listen cinder
  bind 192.168.1.150:8776 transparent
  bind 172.16.0.10:8776 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8776
check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8776
check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8776
check fall 5 inter 2000 rise 2

listen glance_api
  bind 192.168.1.150:9292 transparent
  bind 172.18.0.10:9292 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /healthcheck
  server overcloud-controller-0.internalapi.localdomain 172.18.0.17:9292
check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.18.0.15:9292
check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.18.0.16:9292
check fall 5 inter 2000 rise 2

listen gnocchi

```

```
bind 192.168.1.150:8041 transparent
bind 172.16.0.10:8041 transparent
mode http
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8041
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8041
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8041
check fall 5 inter 2000 rise 2

listen haproxy.stats
bind 10.200.0.6:1993 transparent
mode http
stats enable
stats uri /
stats auth admin:PnDD32EzdVCf73CpjHhFGHZdV

listen heat_api
bind 192.168.1.150:8004 transparent
bind 172.16.0.10:8004 transparent
mode http
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
timeout client 10m
timeout server 10m
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8004
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8004
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8004
check fall 5 inter 2000 rise 2

listen heat_cfn
bind 192.168.1.150:8000 transparent
bind 172.16.0.10:8000 transparent
mode http
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
timeout client 10m
timeout server 10m
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8000
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8000
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8000
check fall 5 inter 2000 rise 2

listen horizon
bind 192.168.1.150:80 transparent
bind 172.16.0.10:80 transparent
mode http
```

```

    cookie SERVERID insert indirect nocache
    option forwardfor
    option httpchk
    server overcloud-controller-0.internalapi.localdomain 172.16.0.13:80
check cookie overcloud-controller-0 fall 5 inter 2000 rise 2
    server overcloud-controller-1.internalapi.localdomain 172.16.0.14:80
check cookie overcloud-controller-0 fall 5 inter 2000 rise 2
    server overcloud-controller-2.internalapi.localdomain 172.16.0.15:80
check cookie overcloud-controller-0 fall 5 inter 2000 rise 2

listen keystone_admin
    bind 192.168.24.15:35357 transparent
    mode http
    http-request set-header X-Forwarded-Proto https if { ssl_fc }
    http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
    option httpchk GET /v3
    server overcloud-controller-0.ctlplane.localdomain 192.168.24.9:35357
check fall 5 inter 2000 rise 2
    server overcloud-controller-1.ctlplane.localdomain 192.168.24.8:35357
check fall 5 inter 2000 rise 2
    server overcloud-controller-2.ctlplane.localdomain 192.168.24.18:35357
check fall 5 inter 2000 rise 2

listen keystone_public
    bind 192.168.1.150:5000 transparent
    bind 172.16.0.10:5000 transparent
    mode http
    http-request set-header X-Forwarded-Proto https if { ssl_fc }
    http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
    option httpchk GET /v3
    server overcloud-controller-0.internalapi.localdomain 172.16.0.13:5000
check fall 5 inter 2000 rise 2
    server overcloud-controller-1.internalapi.localdomain 172.16.0.14:5000
check fall 5 inter 2000 rise 2
    server overcloud-controller-2.internalapi.localdomain 172.16.0.15:5000
check fall 5 inter 2000 rise 2

listen mysql
    bind 172.16.0.10:3306 transparent
    option tcpka
    option httpchk
    stick on dst
    stick-table type ip size 1000
    timeout client 90m
    timeout server 90m
    server overcloud-controller-0.internalapi.localdomain 172.16.0.13:3306
backup check inter 1s on-marked-down shutdown-sessions port 9200
    server overcloud-controller-1.internalapi.localdomain 172.16.0.14:3306
backup check inter 1s on-marked-down shutdown-sessions port 9200
    server overcloud-controller-2.internalapi.localdomain 172.16.0.15:3306
backup check inter 1s on-marked-down shutdown-sessions port 9200

listen neutron
    bind 192.168.1.150:9696 transparent
    bind 172.16.0.10:9696 transparent
    mode http

```

```
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:9696
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:9696
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:9696
check fall 5 inter 2000 rise 2

listen nova_metadata
bind 172.16.0.10:8775 transparent
option httpchk
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8775
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8775
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8775
check fall 5 inter 2000 rise 2

listen nova_novncproxy
bind 192.168.1.150:6080 transparent
bind 172.16.0.10:6080 transparent
balance source
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option tcpka
timeout tunnel 1h
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6080
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6080
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6080
check fall 5 inter 2000 rise 2

listen nova_osapi
bind 192.168.1.150:8774 transparent
bind 172.16.0.10:8774 transparent
mode http
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8774
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8774
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8774
check fall 5 inter 2000 rise 2

listen nova_placement
bind 192.168.1.150:8778 transparent
bind 172.16.0.10:8778 transparent
mode http
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
```

```

server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8778
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8778
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8778
check fall 5 inter 2000 rise 2

listen panko
bind 192.168.1.150:8977 transparent
bind 172.16.0.10:8977 transparent
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
option httpchk
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8977
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8977
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8977
check fall 5 inter 2000 rise 2

listen redis
bind 172.16.0.13:6379 transparent
balance first
option tcp-check
tcp-check send AUTH\ V2EgUh2pvkr8VzU6yuE4XHsr9\r\n
tcp-check send PING\r\n
tcp-check expect string +PONG
tcp-check send info\ replication\r\n
tcp-check expect string role:master
tcp-check send QUIT\r\n
tcp-check expect string +OK
server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6379
check fall 5 inter 2000 rise 2
server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6379
check fall 5 inter 2000 rise 2
server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6379
check fall 5 inter 2000 rise 2

listen swift_proxy_server
bind 192.168.1.150:8080 transparent
bind 172.18.0.10:8080 transparent
option httpchk GET /healthcheck
timeout client 2m
timeout server 2m
server overcloud-controller-0.storage.localdomain 172.18.0.17:8080 check
fall 5 inter 2000 rise 2
server overcloud-controller-1.storage.localdomain 172.18.0.15:8080 check
fall 5 inter 2000 rise 2
server overcloud-controller-2.storage.localdomain 172.18.0.16:8080 check
fall 5 inter 2000 rise 2

```

### **/etc/corosync/corosync.conf file (コントローラーノード)**

このファイルは、クラスターのインフラストラクチャーを定義し、全コントローラーノード上で利用できます。

```

totem {
  version: 2
  cluster_name: tripleo_cluster
  transport: udpu
  token: 10000
}

nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }

  node {
    ring0_addr: overcloud-controller-1
    nodeid: 2
  }

  node {
    ring0_addr: overcloud-controller-2
    nodeid: 3
  }
}

quorum {
  provider: corosync_votequorum
}

logging {
  to_logfile: yes
  logfile: /var/log/cluster/corosync.log
  to_syslog: yes
}

```

### **/etc/ceph/ceph.conf (Ceph ノード)**

このファイルには、Ceph の高可用性設定が記載されています。これには、モニタリングするホストにホスト名、IP アドレスが含まれます。

```

[global]
osd_pool_default_pgp_num = 128
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-
1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx

```



```
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```