Red Hat OpenStack Platform 8

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

Last Updated: 2018-09-07

Red Hat OpenStack Platform 8 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

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 8 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 8 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 8 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 8 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

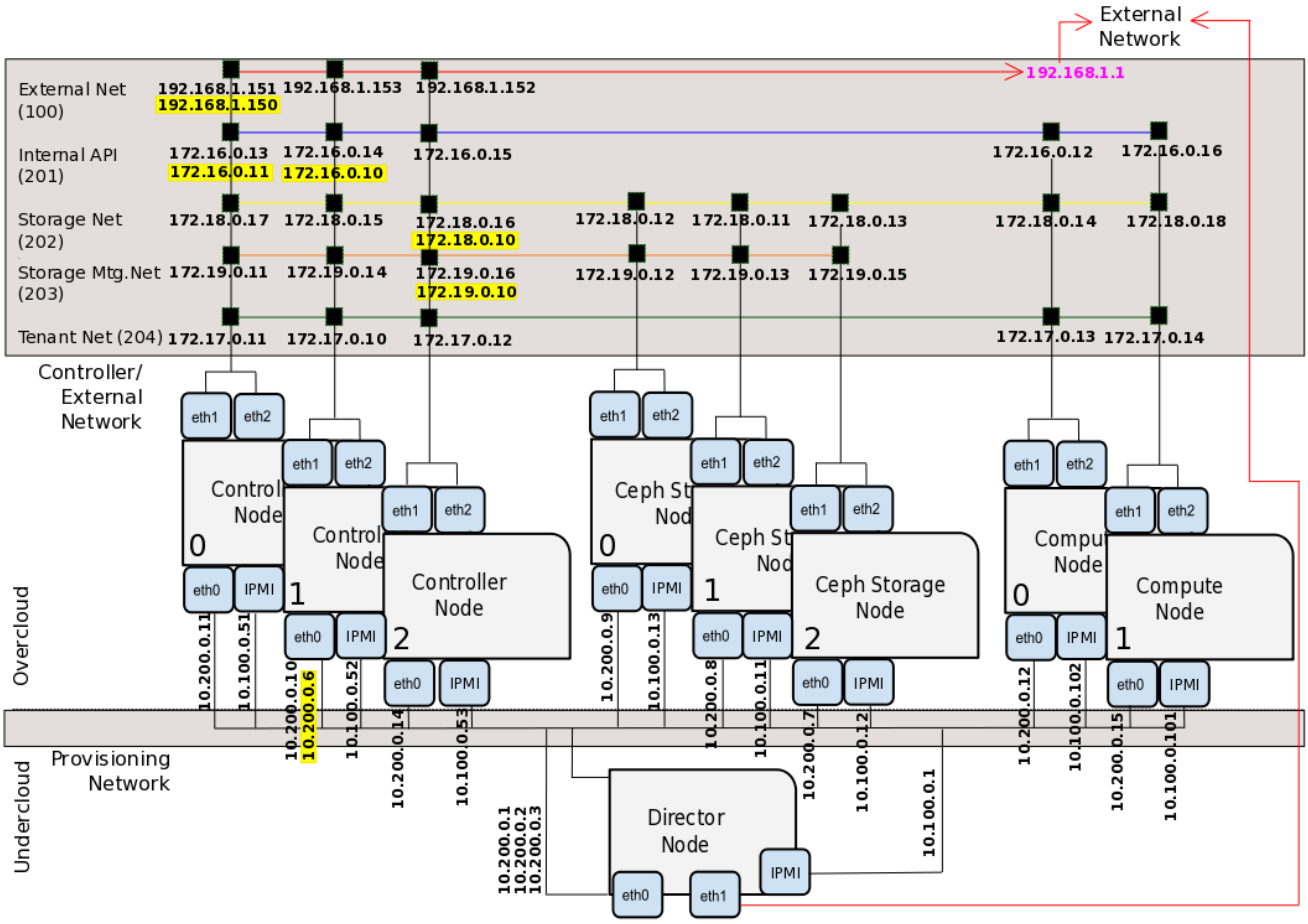
第1章 概要

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。

- 『Red Hat Ceph Storage for the Overcloud』
- 『director のインストールと使用方法』

図1.1「director を使用してデプロイした OpenStack HA 環境」は、本ガイドに記載の HA 機能のテスト専用に構築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 8 HA 環境の構築*」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



HA デプロイメントでは、Pacemaker または HAProxy によって全 OpenStack サービスが起動/管理される必要があります。これには、関連するサービスおよび依存関係にあるサービスもすべて含まれます。

たとえば、**httpd** サービスは **openstack-dashboard** に必要です。そのため、HA 環境では **httpd** を手動で (たとえば、**pcs** ではなく **systemctl** を使用して) **起動/有効化してはなりません**。HA デプロイメントにおけるコロケーションや依存関係の問題の多くは、サービスが Pacemaker または HAProxy 以外で管理され ていることが原因です。

この問題を回避するには、HA デプロイメントをすべて director でオーケストレーションを行ってください。director が使用するテンプレートと puppet のモジュールにより、全サービスが正しく設定/起動されます。さらに、HA の問題をトラブルシューティングする際には、可能な場合には HA フレームワークを介してサービスと常に対話を行ってください。

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

Red Hat OpenStack Platform 8 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、コンピュート、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 8 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、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 はサービスの再起動、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 8 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することができます。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
```

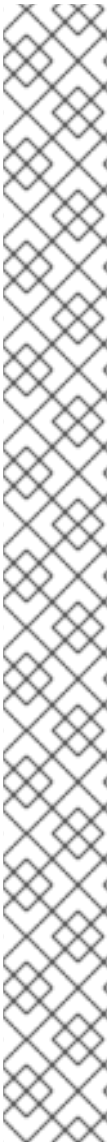
同様に、オーバークラウドと対話するには、以下のコマンドを実行します。

```
$ 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ついての詳しい説明は、「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
$ nova 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	アドレス
overcloud-controller-0	**10.200.0.11**
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します。

```
$ source ~stack/overcloudrc
$ ssh heat-admin@10.200.0.11
```

コントローラー、コンピュート、またはストレージシステムにログインした後には、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 の **pcs status** コマンドからの出力には、Pacemaker の一般情報、仮想 IP アドレス、サービス、その他の Pacemaker 情報が含まれます。

4.1. PACEMAKER の一般情報

pcs status の出力の最初の部分には、クラスターの名前、クラスターの最近の変更、現在の DC、クラスター内のノード数、クラスターで設定されたリソース数、クラスター内のノードが表示されます。

```
$ sudo pcs status
    Cluster name: tripleo_cluster
    Last updated: Mon Oct  5 13:42:37 2015
    Last change: Mon Oct  5 13:03:06 2015
    Stack: corosync
    Current DC: overcloud-controller-1 (2) - partition with quorum
    Version: 1.1.12-a14efad
    3 Nodes configured
    115 Resources configured
    Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]

    Full list of resources:
...
```

sudo pcs status からの最初の出力では、クラスターの名前は **tripleo_cluster** で、3 つのノード (overcloud-controller-0、-1、-2) で構成されていることが分かります。これら 3 つのノードはすべて現在オンライン状態です。

また、名前が **tripleo_cluster** のクラスター内で管理されるように設定されたリソースは、システムのデプロイ方法により変化する可能性があります。この例では、リソースは 115 個あります。

pcs status からの出力の次の部分では、どのリソースが開始されたのか (IP アドレス、サービスなど)、さらにそれらのリソースはどのコントローラーノードで実行されているのかが分かります。以下の項では、その出力例を紹介します。

Pacemaker についてのさらに詳しい情報は、以下のドキュメントを参照してください。

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

4.2. 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際に使用する仮想 IP アドレスを設定します。その IP アドレスが割り当てられたコントローラーノードがダウンした場合には、この IP アドレスは別のコントローラーに再度割り当てられます。この例では、各コントローラー (overcloud-controller-0、-1 など) が現在、特定の仮想 IP アドレスをリッスンするように設定されていることが分かります。

```
 ip-192.168.1.150 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
 ip-10.200.0.6 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
 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
```

各 IP アドレスは最初に、特定のコントローラー (例: **overcloud-controller-0** 上で **192.168.1.150** が起動) にアタッチされる点に注意してください。ただし、そのコントローラーが停止した場合には、その IP アドレスは、このクラスター内の別のコントローラーに再割り当てされます。上記の IP アドレスについての説明と、それらの IP アドレスが最初にどのように割り当てられたのかを以下に記載します。

- **192.168.1.150**: パブリック IP アドレス (**network-environment.yaml** の ExternalAllocationPools から割り当て)
- **10.200.0.6**: コントローラーの仮想 IP アドレス (**undercloud.conf** 内に dhcp_start および dhcp_end range 範囲セットの一部は 10.200.0.5-10.200.0.24 に設定)
- **172.16.0.10**: コントローラー上の OpenStack API サービスへのアクセスを提供する IP アドレス (**network-environment.yaml** の InternalApiAllocationPools から割り当て)
- **172.16.0.11**: コントローラー上の Redis サービスへのアクセスを提供する IP アドレス (**network-environment.yaml** の InternalApiAllocationPools から割り当て)
- **172.18.0.10**: Glance API および Swift プロキシーサービスへのアクセスを提供するストレージ仮想 IP アドレス (**network-environment.yaml** の StorageAllocationPools から割り当て)
- **172.19.0.10**: ストレージ管理へのアクセスを提供する IP アドレス (**network-environment.yaml** の StorageMgmtAlloctionPools から割り当て)

pcs コマンドを使用して、Pacemaker に設定された特定の IPaddr2 アドレスに関する詳細を表示することができます。たとえば、特定の仮想 IP アドレスに関するタイムアウトおよびその他の付属情報を表示するには、IPaddr2 のいずれかで以下を入力します。

```
$ 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します。

```
$ 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
    tcp  0  0 192.168.1.150:6080      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:9696      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8000      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8003      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8004      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8773      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8774      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:5000      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8776      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8777      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:9292      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:8080      0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:80        0.0.0.0:*  LISTEN      4333/haproxy
    tcp  0  0 192.168.1.150:35357     0.0.0.0:*  LISTEN      4333/haproxy
    udp  0  0 192.168.1.150:123       0.0.0.0:*              459/ntpd
    ...
    tcp  0  0 0.0.0.0:22              0.0.0.0:*  LISTEN      2471/sshd
    tcp  0  0 0.0.0.0:4567            0.0.0.0:*  LISTEN      10064/mysqld
    ...
    udp  0  0 0.0.0.0:51475           0.0.0.0:*              545/dhclient
    udp  0  0 0.0.0.0:123             0.0.0.0:*              459/ntpd
    udp  0  0 0.0.0.0:161             0.0.0.0:*              1633/snmpd
    ...
```

ip コマンドから、vlan100 インターフェースが 192.168.1.150 および 192.168.1.151 IPv4 アドレスの両方をリッスンしていることが分かります。**netstat** コマンドの出力では、192.168.1.150 インターフェースでリッスンするプロセスすべてが表示されます。ntpd プロセス (ポート 123 をリッスン) 以外に、haproxy プロセスだけが 192.168.1.150 を専用でリッスンするプロセスです。また、すべてのローカルアドレス (0.0.0.0) をリッスンするプロセスも 192.168.1.150 (sshd、mysqld、dhclient、ntpd など) 経由で利用可能である点に注意してください。

netstat の出力で表示されたポート番号により、haproxy がリッスンするサービスが具体的にどれなのかを特定しやすくなります **/etc/haproxy/haproxy.cfg** ファイルの内容から、これらのポート番号がどのサービスを表しているかを確認してください。以下にいくつか例を示します。

- **TCP ポート 6080**: nova_novncproxy
- **TCP ポート 9696**: neutron
- **TCP ポート 8000**: heat_cfn
- **TCP ポート 8003**: heat_cloudwatch
- **TCP ポート 80**: horizon

今回は、**haproxy.cfg** 内には、3 つのコントローラーすべてで 192.168.1.150 を特にリッスンしている 14 のサービスがあります。ただし、現在、実際に 192.168.1.150 を外部でリッスンしているのは controller-0 のみなので、controller-0 が停止した場合には、HAProxy が別のコントローラーに再割当てする必要があるのは 192.168.1.150 のみで、全サービスがすでに実行中となります。

4.3. PACEMAKER で設定された OPENSTACK サービス

大半のサービスは、**Clone Set** リソース (または **clones**) として設定されます。この設定では、リソースは各コントローラーで同じ方法で起動され、各コントローラーで常時実行されます。サービスは、複

数のノードでアクティブな状態である必要がある場合にクローンされます。そのため、クローンが可能なのは、複数のノードで同時にアクティブにすることが可能なサービス (**cluster-aware services**) のみとなります。

その他のサービスは、**Multi-state** リソースとして設定されます。**Multi-state** リソースは、通常の **Clone Set** リソースとは異なり、特別なタイプのクローンです。**Multi-state** リソースは **マスター** または **スレーブ** のいずれかの状態が可能です。インスタンスの起動時には、**スレーブ** の状態である必要があります。それ以外には、いずれの状態名も特別な意味はありませんが、これらの状態により、同じサービスのクローンが異なるルールまたは制約に従って実行することを可能となります。

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が、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 アドレスをリッスンしていない場合もあります。

Clone Set リソース (clones)

pcs status からのクローン設定は以下のようになります。

```
Clone Set: haproxy-clone [haproxy]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: mongod-clone [mongod]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: rabbitmq-clone [rabbitmq]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: memcached-clone [memcached]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-nova-scheduler-clone [openstack-nova-scheduler]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-l3-agent-clone [neutron-l3-agent]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-ceilometer-alarm-notifier-clone [openstack-
ceilometer-alarm-notifier]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-heat-engine-clone [openstack-heat-engine]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-ceilometer-api-clone [openstack-ceilometer-api]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-metadata-agent-clone [neutron-metadata-agent]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-ovs-cleanup-clone [neutron-ovs-cleanup]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-netns-cleanup-clone [neutron-netns-cleanup]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-heat-api-clone [openstack-heat-api]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
```

```
controller-2 ]
 Clone Set: openstack-cinder-scheduler-clone [openstack-cinder-scheduler]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-nova-api-clone [openstack-nova-api]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-heat-api-cloudwatch-clone [openstack-heat-api-
cloudwatch]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-ceilometer-collector-clone [openstack-ceilometer-
collector]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-keystone-clone [openstack-keystone]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-nova-consoleauth-clone [openstack-nova-consoleauth]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-glance-registry-clone [openstack-glance-registry]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-c
openstack-cinder-volume
 Clone Set: openstack-ceilometer-notification-clone [openstack-ceilometer-
notification]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-cinder-api-clone [openstack-cinder-api]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-dhcp-agent-clone [neutron-dhcp-agent]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-glance-api-clone [openstack-glance-api]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-openvswitch-agent-clone [neutron-openvswitch-agent]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-nova-novncproxy-clone [openstack-nova-novncproxy]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: delay-clone [delay]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: neutron-server-clone [neutron-server]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: httpd-clone [httpd]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-ceilometer-central-clone [openstack-ceilometer-
central]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
```

```
 Clone Set: openstack-ceilometer-alarm-evaluator-clone [openstack-
ceilometer-alarm-evaluator]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 Clone Set: openstack-heat-api-cfn-clone [openstack-heat-api-cfn]
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
 openstack-cinder-volume  (systemd:openstack-cinder-volume):  Started
overcloud-controller-0
 Clone Set: openstack-nova-conductor-clone [openstack-nova-conductor]
openstack-cinder-volume
     Started: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
```

Clone Set リソースごとに、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
- 実際のサービス名
- サービスが起動または停止されるコントローラー

Clone Set を使用すると、サービスは、全コントローラー上で同じ方法で起動されるようになります。特定のクローンサービス (haproxy サービスなど) に関する詳細を表示するには、**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
 Clone: haproxy-clone
  Resource: haproxy (class=systemd type=haproxy)
   Operations: start interval=0s timeout=60s (haproxy-start-timeout-60s)
               monitor interval=60s (haproxy-monitor-interval-60s)
$ sudo systemctl status haproxy
haproxy.service - Cluster Controlled haproxy
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; disabled)
  Drop-In: /run/systemd/system/haproxy.service.d
           └─50-pacemaker.conf
   Active: active (running) since Tue 2015-10-06 08:58:49 EDT; 1h 52min
ago
 Main PID: 4215 (haproxy-systemd)
   CGroup: /system.slice/haproxy.service
           ├─4215 /usr/sbin/haproxy-systemd-wrapper -f
/etc/haproxy/haproxy.cfg -p /run/haproxy.pid
           ├─4216 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p
/run/haproxy.pid -Ds
           └─4217 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p
/run/haproxy.pid -Ds
```

haproxy-clone の例では、HAProxy のリソース設定を確認できます。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 Pacemaker のクローンサービスとして HAProxy を設定して HAProxy 自体も高可用性に保ちます。

この出力では、リソースが **haproxy** という名前の **systemd** サービスである点に注意してください。また、開始の間隔やタイムアウトの値、監視の間隔なども記載されています。**systemctl status** コマンドでは、**haproxy** がアクティブであることが分かります。**haproxy** サービスの実際に実行中のプロセス

は、出力の最後に一覧表示されます。全コマンドラインが表示されているため、設定ファイル (**haproxy.cfg**) および PID ファイル (**haproxy.pid**) がこのコマンドと関連付けられていることが分かります。

任意の **Clone Set** リソースに同じコマンドを実行して、アクティビティーの現在のレベルやサービスが実行するコマンドの詳細を表示します。Pacemaker によって制御される **systemd** サービスは、**systemd** では **disabled** に設定されている点に注意してください。これは、サービスの起動または停止時には、システムのブートプロセスではなく Pacemaker により制御させるようにする必要があるためです。

Clone Set リソースについての詳しい情報は、『High Availability Add-On リファレンス』で「リソースのクローン」の項を参照してください。

マルチステートのリソース (マスター/スレーブ)

Galera および Redis のサービスは、**マルチステート** リソースとして実行されます。これらの 2 種類のサービスの **pcs status** 出力例を以下に示します。

```
[...]
Master/Slave Set: galera-master [galera]
     Masters: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]

Master/Slave Set: redis-master [redis]
     Masters: [ overcloud-controller-2 ]
     Slaves: [ overcloud-controller-0 overcloud-controller-1 ]
[...]
```

galera-master リソースでは、3 つのコントローラーはすべて Gelera マスターとして実行されます。**redis-master** リソースで **overcloud-controller-2** がマスターとして、残りの 2 つのコントローラーがスレーブとして実行されています。これは、現在 **galera** サービスが 3 つのコントローラーで単一の制約セットに従って実行されている一方で、**redis** はマスターまたはスレーブコントローラー上の異なる制約に従っている可能性があることを意味します。

マルチステート リソースについての詳しい情報は、『High Availability Add-On リファレンス』で「多状態のリソース: 複数モードのリソース」を参照してください。

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

いずれかのリソースに何らかのエラーが発生した場合には、**pcs status** の出力の **Failed actions** の見出しの下に一覧表示されます。**controller-0** で **httpd** サービスが停止した場合の例を以下に示します。

```
Failed actions:
    httpd_monitor_60000 on overcloud-controller-0 'not running' (7): call=
openstack-cinder-volume        (systemd:openstack-cinder-volume):
Started overcloud-controller-0
190, status=complete, exit-reason='none', last-rc-change='Thu Oct  8
10:12:32 2015', queued=0ms, exec=0ms
```

この場合に必要な操作は、**systemd** service **httpd** を再起動するのみです。その他の場合には、問題を特定して修正し、リソースをクリーンアップする必要があります。詳しくは、「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**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。

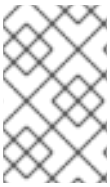
```
$ sudo pcs stonith show --full
 Resource: my-ipmilan-for-controller-0 (class=stonith type=fence_ipmilan)
  Attributes: pcmk_host_list=overcloud-controller-0 ipaddr=10.100.0.51
login=admin passwd=abc lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-0-monitor-
interval-60s)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

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 サービス自体を高可用性サービスとして管理します (**haproxy-clone**)。



注記

HAProxy の設定を検証する方法については、「How can I verify my haproxy.cfg is correctly configured to load balance openstack services?」の KCS ソリューションを参照してください。

Red Hat OpenStack Platform 8 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。これは **/etc/haproxy/haproxy.cfg** ファイルでこれらのサービスを設定することによって実現されます。このファイルのサービスごとに、以下の設定を確認できます。

- **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 7 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 **ceilometer** の listen サービスの設定例を以下に示します。

```
listen ceilometer
  bind 172.16.0.10:8777
  bind 192.168.1.150:8777
  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
```

ceilometer サービスの HAProxy 設定に関するこの例では、**ceilometer** サービスを提供する (172.16.0.10 および 192.168.1.150 上のポート 8777) IP アドレスおよびポートが分かります。172.16.0.10 のアドレスは、オーバークラウド内で使用する、内部 API ネットワーク (VLAN201) 上の仮想 IP アドレスで、192.168.1.150 は、オーバークラウドの外部から API ネットワークにアクセスできるようにする、外部ネットワーク (VLAN100) 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

ceilometer	cinder	glance_api	glance_registry
haproxy.stats	heat_api	heat_cfn	heat_cloudwatch
horizon	keystone_admin	keystone_public	mysql
neutron	nova_ec2	nova_metadata	nova_novncproxy

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 **/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 /
```

上記の場合には、HAProxy Stats を表示するには、Web ブラウザーでアドレスを **10.200.0.6:1993** に指定します。

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 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/スレーブ設定のリソースを使用して Galera サービスを実行します。**pcs status** を使用して、**galera-master** が実行されているかどうか、またその場合にはどのコントローラー上で実行されているのかを確認することができます。

```
Master/Slave Set: galera-master [galera]
     Masters: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ]
```

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** から開始してください。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。[1]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。

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 -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

以下の項では、各ステータスを調べる方法について説明します。

6.1. データベースクラスターの整合性の調査

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から開始してください。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 -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 -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と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
wsrep_cluster_state_uuid	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。
wsrep_cluster_size	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、接続を失ったことになります。
wsrep_cluster_conf_id	クラスターの変更回数	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。 異なる **wsrep_cluster_conf_id** を報告するノードがある場合には、**wsrep_cluster_status** の値をチェックして、クラスター (**Primary**) への書き込みができるかどうかを確認してください。

変数	概要	説明
wsrep_cluster_status	Primary コンポーネントのステータス	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、**wsrep_cluster_status** は **Primary** 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。

6.2. データベースクラスターノードの調査

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、クラスターのチェック (「データベースクラスターの整合性の調査」に記載) と同様の方法で確認することができます。

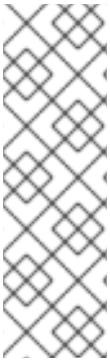
```
$ sudo mysql -B -e "SHOW GLOBAL STATUS LIKE 'VARIABLE';"
```

同様に、**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
wsrep_ready	ノードがクエリーを受け入れる機能	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、**wsrep_ready** は **ON** のはずです。
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、**wsrep_connected** は **ON** のはずです。

変数	概要	説明
wsrep_local_state_comment	ノードの状態	ノードの状態の概要を示します。ノードがクラスターにまだ書き込みができる場合 (すなわち、**wsrep_cluster_status** が **Primary** の場合。「データベースクラスターの整合性の調査」 を参照) には、**wsrep_local_state_comment** の通常の値は **Joining**、**Waiting on SST**、**Joined**、**Synced**、**Donor** です。 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、**wsrep_local_state_comment** は **Initialized** 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ーの整合性の調査」 を参照してください。

wsrep_connected が **OFF** の場合には、そのノードは、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。

6.3. データベースレプリケーションのパフォーマンスの調査

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もできます。「データベースクラスターノードの調査」および「データベースクラスターの整合性の調査」に記載したように、この操作には各ノードの **wsrep_** データベース変数が必要です。

```
$ sudo mysql -B -e "SHOW STATUS LIKE 'VARIABLE';"
```

同様に、**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
wsrep_local_recv_queue_avg	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

変数	概要
wsrep_local_send_queue_avg	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
wsrep_local_recv_queue_min and wsrep_local_recv_queue_max	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
wsrep_flow_control_paused	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
wsrep_cert_deps_distance	並行して適用可能なシーケンス番号 (**seqno**) の最小値と最大値の幅の平均 (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。

異なる値とベンチマークについての説明は以下の一覧を確認してください。

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。

スロットルおよび一時停止の場合には、**wsrep_cert_deps_distance** 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。その後に、**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。

Configuring a higher **wsrep_slave_threads** の値を高くすると、スロットルと一時停止を軽減することができます。たとえば、**wsrep_cert_deps_distance** の読み取った値が **20** の場合は、**wsrep_slave_threads** を **2** からその倍の **4** に指定すると、その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** はそのノードの CPU コア数以下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、どのリソースと一緒に起動する必要があるのかなどの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。

pcs constraint show コマンドは、場所 (特定のホスト上でのみ実行可能)、順序 (起動前に他のリソースが有効化されているかによって左右される)、コロケーション (他のリソースと同じ場所に配置されている必要がある) などによりリソースがどのように制約されているかを表示します。コントローラーノードで実行した **pcs constraint show** の出力を途中省略して以下に記載します。

```
$ sudo pcs constraint show
Location Constraints:
  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-clone (kind:Optional)
  start ip-10.200.0.6 then start haproxy-clone (kind:Optional)
  start ip-172.19.0.10 then start haproxy-clone (kind:Optional)
  start ip-192.168.1.150 then start haproxy-clone (kind:Optional)
  start ip-172.16.0.11 then start haproxy-clone (kind:Optional)
  start ip-172.18.0.10 then start haproxy-clone (kind:Optional)
  start mongod-clone then start openstack-ceilometer-central-clone
(kind:Mandatory)
  start openstack-glance-registry-clone then start openstack-glance-api-
clone (kind:Mandatory)
  start openstack-heat-api-clone then start openstack-heat-api-cfn-clone
(kind:Mandatory)
  start delay-clone then start openstack-ceilometer-alarm-evaluator-clone
(kind:Mandatory)

...

Colocation Constraints:
  ip-172.16.0.10 with haproxy-clone (score:INFINITY)
  ip-172.18.0.10 with haproxy-clone (score:INFINITY)
  ip-10.200.0.6 with haproxy-clone (score:INFINITY)
  ip-172.19.0.10 with haproxy-clone (score:INFINITY)
  ip-172.16.0.11 with haproxy-clone (score:INFINITY)
  ip-192.168.1.150 with haproxy-clone (score:INFINITY)
  openstack-glance-api-clone with openstack-glance-registry-clone
(score:INFINITY)
  openstack-cinder-volume with openstack-cinder-scheduler-clone
(score:INFINITY)
  neutron-dhcp-agent-clone with neutron-openvswitch-agent-clone
(score:INFINITY)

...
```

この出力には、以下の 3 つの主要なセクションが表示されています。

Location Constraints

このセクションには、リソースの割り当て先の制約は特にないことが表示されています。ただし、出力では、 **ipmilan** リソースが各コントローラー上で無効化されていることが分かります。そのため、この点に関してさらに調査する必要があります。

Ordering Constraints

このセクションでは、仮想 IP アドレスリソース (IPaddr2) は HAProxy より前に起動するように設定されている点に注目してください。**mongod-clone** を **openstack-ceilometer-central-clone** よりも前に起動することや、**openstack-glance-registry-clone** を **openstack-glance-api-clone** よりも前に起動することなど、必須の Ordering Constraints も多数あります。これらの制約を把握することによって、サービス間の依存関係を理解することができます。言い換えると、問題のあるサービスまたは他のリソースを修正できるようにするには、どのような依存関係を設定する必要があるかが分かります。

Colocation 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。たとえば、特定の IP アドレスが **haproxy-clone** リソースに関連づけられています。また、**openstack-glance-api-clone** リソースは **openstack-glance-registry-clone** リソースと同じホストに配置する必要があります。

7.1. コントローラー上のリソースの問題修正

失敗したアクションは、**pcs status** コマンドで表示します。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試してください。

エラーが発生した個別リソースの問題を特定するには、「*7章HA コントローラーリソースの調査と修正*」で説明した **Ordering Constraints** を確認します。エラーの発生したリソースが依存するリソースすべてが稼働していることを確認し、下から順番に修正していきます。

エラーが発生したリソースとリソースが実行されるコントローラーの名前を指定して、コントローラーにログインして問題のデバッグすることができます。エラーが発生したリソースが **systemd** サービスの場合は (例: **openstack-ceilometer-api**)、**systemctl** でそのステータスを確認し、**journalctl** でジャーナルメッセージを検索することが可能です。以下に例を示します。

```
$ sudo systemctl status openstack-ceilometer-api
openstack-ceilometer-api.service - Cluster Controlled openstack-
ceilometer-api
   Loaded: loaded (/usr/lib/systemd/system/openstack-ceilometer-
api.service; disabled)
```

```
  Drop-In: /run/systemd/system/openstack-ceilometer-api.service.d
           └─50-pacemaker.conf
   Active: active (running) since Thu 2015-10-08 13:30:44 EDT; 1h 4min ago
 Main PID: 17865 (ceilometer-api)
   CGroup: /system.slice/openstack-ceilometer-api.service
           └─17865 /usr/bin/python /usr/bin/ceilometer-api --logfile
/var/log/ceilometer/api.log

Oct 08 13:30:44 overcloud-controller-2.localdomain systemd[1]: Starting
Cluster Controlled openstack-ceilo.....
Oct 08 13:30:44 overcloud-controller-2.localdomain systemd[1]: Started
Cluster Controlled openstack-ceilom...i.
Oct 08 13:30:49 overcloud-controller-2.localdomain ceilometer-api[17865]:
/usr/lib64/python2.7/site-package....
$ sudo journalctl -u openstack-ceilometer-api
-- Logs begin at Thu 2015-10-01 08:57:25 EDT, end at Thu 2015-10-08
14:40:18 EDT. --
Oct 01 11:22:41 overcloud-controller-2.localdomain systemd[1]: Starting
Cluster Controlled openstack...
Oct 01 11:22:41 overcloud-controller-2.localdomain systemd[1]: Started
Cluster Controlled openstack-ceilometer-api...
Oct 01 11:22:52 overcloud-controller-2.localdomain ceilometer-api[8918]:
/usr/lib64/python2.7/...
```

エラーが発生したリソースを修正した後には、**pcs resource cleanup** コマンドを実行すると、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。たとえば、**httpd-clone** リソースの問題を修正するには、以下のコマンドを実行します。

```
$ sudo pcs resource cleanup httpd-clone
  Resource: httpd-clone 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 8 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かを確認するには、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。

第9章 HA コンピュートノードの調査

コンピュートノードが停止した場合には、Pacemaker はそのノードで問題が発生したサービスの再起動を試行します。この際には、**neutron-ovs-agent**、 **ceilometer-compute**、**nova-compute** の順でサービスが起動されます。Swift ACO ノードで問題が発生している場合には、Swift サービスの再起動は、**swift-fs**、**swift-object**、**swift-container**、**swift-account** の順で行われます。Pacemaker は、これらのサービスの起動に失敗した場合には、コンピュートノードをフェンスします。

付録A RED HAT OPENSTACK PLATFORM 8 HA 環境の構築

『Red Hat Ceph Storage for the Overcloud』 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	コンピュートノード (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 8 環境の全

コンピューター (オーバークラウド) に接続できる必要があります。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、Pacemaker の **stonith** というツールを使用して決定することができます。詳しい情報は、『director のインストールと使用方法』ガイドの「コントローラーノードのフェンシング」の項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::TripleO::BlockStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/cinder-storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/swift-storage.yaml
  OS::TripleO::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。

/etc/haproxy/haproxy.cfg (コントローラーノード)

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 managed by Puppet
global
  daemon
  group  haproxy
  log  /dev/log local0
  maxconn  10000
  pidfile  /var/run/haproxy.pid
  user  haproxy

defaults
  log  global
  mode  tcp
  option  tcpka
```

```
  option  tcplog
  retries  3
  timeout  http-request 10s
  timeout  queue 1m
  timeout  connect 10s
  timeout  client 1m
  timeout  server 1m
  timeout  check 10s

listen ceilometer
  bind 172.16.0.10:8777
  bind 192.168.1.150:8777
  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

listen cinder
  bind 172.16.0.10:8776
  bind 192.168.1.150:8776
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8776 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8776 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8776 check fall 5 inter 2000
rise 2

listen glance_api
  bind 172.18.0.10:9292
  bind 192.168.1.150:9292
  option httpchk GET /
  server overcloud-controller-0 172.18.0.17:9292 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.18.0.15:9292 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.18.0.16:9292 check fall 5 inter 2000
rise 2

listen glance_registry
  bind 172.16.0.10:9191
  server overcloud-controller-0 172.16.0.13:9191 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:9191 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:9191 check fall 5 inter 2000
rise 2

listen haproxy.stats
  bind 10.200.0.6:1993
  mode http
  stats enable
  stats uri /
```

```
listen heat_api
  bind 172.16.0.10:8004
  bind 192.168.1.150:8004
  mode http
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8004 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8004 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8004 check fall 5 inter 2000
rise 2

listen heat_cfn
  bind 172.16.0.10:8000
  bind 192.168.1.150:8000
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8000 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8000 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8000 check fall 5 inter 2000
rise 2

listen heat_cloudwatch
  bind 172.16.0.10:8003
  bind 192.168.1.150:8003
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8003 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8003 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8003 check fall 5 inter 2000
rise 2

listen horizon
  bind 172.16.0.10:80
  bind 192.168.1.150:80
  cookie SERVERID insert indirect nocache
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:80 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:80 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:80 check fall 5 inter 2000
rise 2

listen keystone_admin
  bind 172.16.0.10:35357
  bind 192.168.1.150:35357
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:35357 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:35357 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:35357 check fall 5 inter 2000
rise 2
```

```
listen keystone_public
  bind 172.16.0.10:5000
  bind 192.168.1.150:5000
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:5000 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:5000 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:5000 check fall 5 inter 2000
rise 2

listen mysql
  bind 172.16.0.10:3306
  option httpchk
  stick on dst
  stick-table type ip size 1000
  timeout client 0
  timeout server 0
  server overcloud-controller-0 172.16.0.13:3306 backup check fall 5 inter
2000 on-marked-down shutdown-sessions port 9200 rise 2
  server overcloud-controller-1 172.16.0.14:3306 backup check fall 5 inter
2000 on-marked-down shutdown-sessions port 9200 rise 2
  server overcloud-controller-2 172.16.0.15:3306 backup check fall 5 inter
2000 on-marked-down shutdown-sessions port 9200 rise 2

listen neutron
  bind 172.16.0.10:9696
  bind 192.168.1.150:9696
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:9696 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:9696 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:9696 check fall 5 inter 2000
rise 2

listen nova_ec2
  bind 172.16.0.10:8773
  bind 192.168.1.150:8773
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8773 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8773 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8773 check fall 5 inter 2000
rise 2

listen nova_metadata
  bind 172.16.0.10:8775
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8775 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8775 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8775 check fall 5 inter 2000
```

```
rise 2

listen nova_novncproxy
  bind 172.16.0.10:6080
  bind 192.168.1.150:6080
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:6080 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:6080 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:6080 check fall 5 inter 2000
rise 2

listen nova_osapi
  bind 172.16.0.10:8774
  bind 192.168.1.150:8774
  option httpchk GET /
  server overcloud-controller-0 172.16.0.13:8774 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:8774 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:8774 check fall 5 inter 2000
rise 2

listen redis
  bind 172.16.0.11:6379
  balance first
  option tcp-check
  tcp-check send info\ replication\r\n
  tcp-check expect string role:master
  timeout client 0
  timeout server 0
  server overcloud-controller-0 172.16.0.13:6379 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.16.0.14:6379 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.16.0.15:6379 check fall 5 inter 2000
rise 2

listen swift_proxy_server
  bind 172.18.0.10:8080
  bind 192.168.1.150:8080
  option httpchk GET /info
  server overcloud-controller-0 172.18.0.17:8080 check fall 5 inter 2000
rise 2
  server overcloud-controller-1 172.18.0.15:8080 check fall 5 inter 2000
rise 2
  server overcloud-controller-2 172.18.0.16:8080 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
secauth: off
cluster_name: tripleo_cluster
transport: udpu
}

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_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
```