Red Hat OpenStack Platform 12

Understanding Red Hat OpenStack Platform High Availability

Understanding, deploying, and managing High Availability in Red Hat OpenStack Platform

Last Updated: 2018-12-05

Red Hat OpenStack Platform 12 Understanding Red Hat OpenStack Platform High Availability

Understanding, deploying, and managing High Availability in Red Hat OpenStack Platform

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

To keep your OpenStack environment up and running efficiently, Red Hat OpenStack Platform 12 Director lets you create configurations that offer high availability and load balancing across all major services in OpenStack. This document describes: A foundational HA setup, created by Red Hat OpenStack Platform 12 Director, that you can use as a reference model for understanding and working with OpenStack HA features. HA features that are used to make various services included in Red Hat OpenStack Platform 12 highly available. Examples of tools for working with and troubleshooting HA features in Red Hat OpenStack Platform 12.

Table of Contents

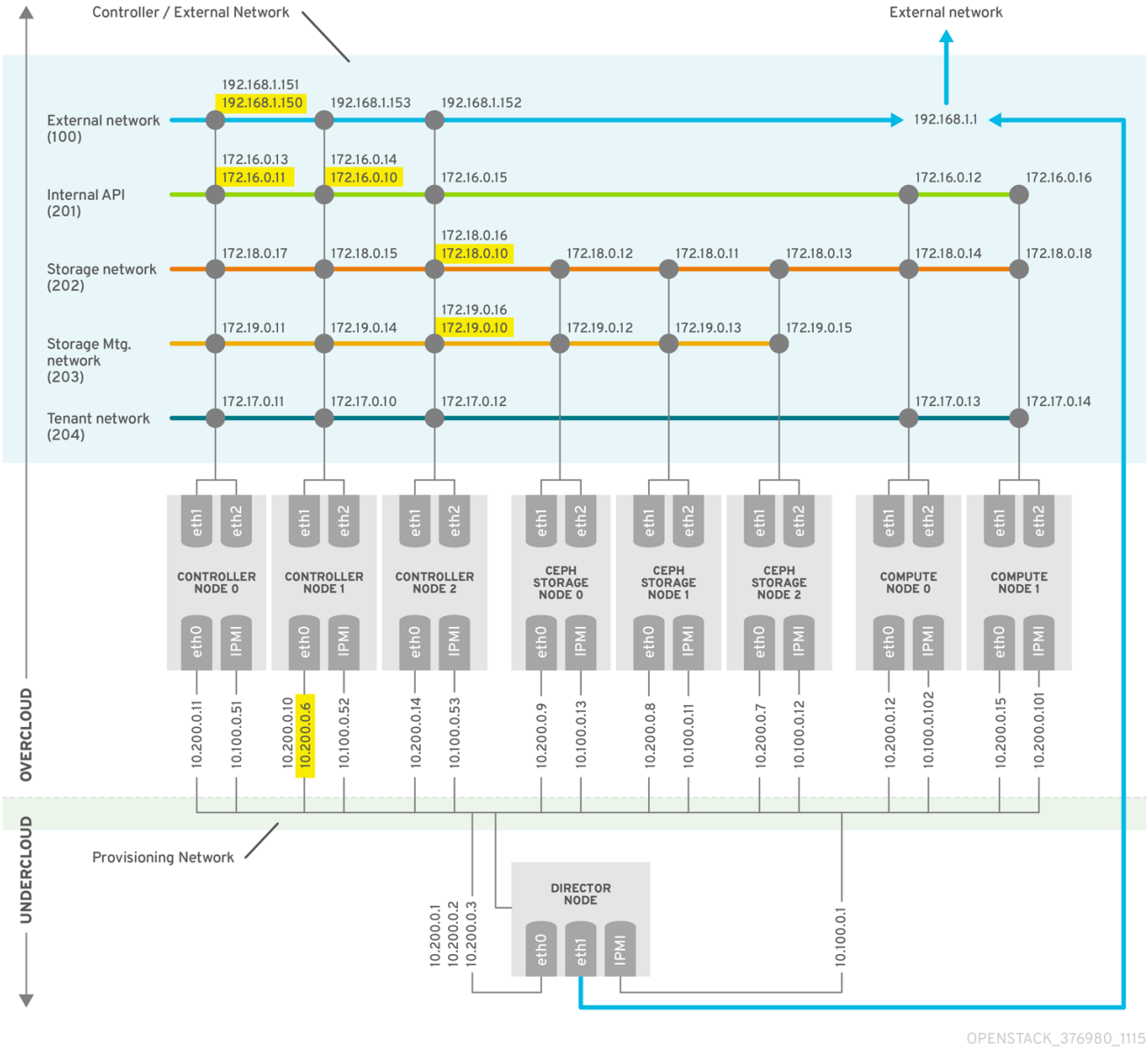
CHAPTER 1. OVERVIEW

The sample HA deployment used for this document was created using the following guides as reference:

- Deploying an Overcloud with Containerized Red Hat Ceph
- Director Installation and Usage

The following figure shows the configuration that was created specifically to test the high availability features described in this document. For details on how to recreate this setup so you can try the steps yourself, see the Appendix A, *Building the Red Hat OpenStack Platform 12 HA Environment* appendix.

Figure 1.1. OpenStack HA environment deployed through director



1.1. MANAGING HIGH AVAILABILITY SERVICES

In a High Availability (HA) deployment, there are four types of services: *core container*, *active-passive*, *systemd* and *plain container*. Core container and active-passive services are launched and managed by Pacemaker. All other services are managed directly by systemd with the **`systemctl`** command or by Docker with the **`docker`** command.

Core container

Core container services include Galera, RabbitMQ, Redis, and HAProxy. These services run on all controller nodes and require specific management and constraints for the start, stop and restart actions.

Active-passive

Active-passive services run on a single controller node at a time, and include services such as **`openstack-cinder-volume`**. Moving an active-passive service must be performed using Pacemaker, which ensures that the correct stop-start sequence is followed.

Systemd and Plain Container

Systemd and Plain Container services are independent services expected to be able to withstand a service interruption, and should not require you to to manually restart any service, such as **`neutron-server.service`**, or any container, such as **`openstack-nova-api-docker`**, if you restart Galera.

When orchestrating your HA deployment entirely with the director, the templates and Puppet modules that are used by the director ensure that all services are configured and launched correctly, particularly for HA. In addition, when troubleshooting HA issues, you need to interact with services using the HA framework, the **`docker`** command, or the **`systemctl`** command.

CHAPTER 2. UNDERSTANDING RED HAT OPENSTACK PLATFORM HIGH AVAILABILITY FEATURES

Red Hat OpenStack Platform employs several technologies to implement high-availability. High availability is offered in different ways for controller, compute, and storage nodes in your OpenStack configuration. To investigate how high availability is implemented, log into each node and run commands, as described in the following sections. The resulting output shows you the high availability services and processes running on each node.

Most of the coverage of high availability (HA) in this document relates to controller nodes. There are two primary HA technologies used on Red Hat OpenStack Platform controller nodes:

- **Pacemaker**: By configuring virtual IP addresses, containers, services, and other features, as resources in a cluster, Pacemaker makes sure that the defined set of OpenStack cluster resources are running and available. When a service or an entire node in a cluster fails, Pacemaker can restart the resource, take the node out of the cluster, or reboot the node. Requests to most of those services is done through HAProxy.

- **HAProxy**: When you configure more than one controller node with the director in Red Hat OpenStack Platform, HAProxy is configured on those nodes to load balance traffic to some of the OpenStack services running on those nodes.

- **Galera**: Red Hat OpenStack Platform uses the MariaDB Galera Cluster to manage database replication.

Highly available services in OpenStack run in one of two modes:

- **Active/active**: In this mode, the same service is brought up on multiple controller nodes with Pacemaker, then traffic can either be distributed across the nodes running the requested service by HAProxy or directed to a particular controller via a single IP address. In some cases, HAProxy distributes traffic to active/active services in a round robin fashion. Performance can be improved by adding more controller nodes.

- **Active/passive**: Services that are not capable of or reliable enough to run in active/active mode are run in active/passive mode. This means that only one instance of the service is active at a time. For Galera, HAProxy uses stick-table options to make sure incoming connections are directed to a single backend service. Galera master-master mode can deadlock when services are accessing the same data from multiple galera nodes at once.

As you begin exploring the high availability services described in this document, keep in mind that the director system (referred to as the undercloud) is itself running OpenStack. The purpose of the undercloud (director system) is to build and maintain the systems that will become your working OpenStack environment. That environment you build from the undercloud is referred to as the overcloud. To get to your overcloud, this document has you log into your undercloud, then choose which Overcloud node you want to investigate.

CHAPTER 3. GETTING INTO YOUR OPENSTACK HA ENVIRONMENT

With the OpenStack HA environment running, log into your director (undercloud) system. Then, become the **stack** user by running:

```
# sudo su - stack
```

From there, you can interact with either the undercloud and overcloud by loading the corresponding environment variables.

To interact with the undercloud, run:

```
$ source ~/stackrc
```

To interact with the overcloud, run:

```
$ source ~/overcloudrc
```

For general information about accessing either the undercloud or the overcloud, see Accessing the Overcloud.

To access and investigate a node, first find out which IP addresses were assigned to them. This involves interacting with the undercloud:

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

NOTE

For reference, the director deployed the following names and addresses in our test environment:

Name	IP Address
overcloud-controller-0	**10.200.0.11**
overcloud-controller-1	10.200.0.10
overcloud-controller-1	10.200.0.6 (virtual 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

In your own test environment, even if you use the same address ranges, the IP addresses assigned to each node might be different.

After you determine the IP addresses of your overcloud nodes, you can run the following command to log into one of those nodes. Doing so involves interacting with the overcloud. For example, to log into **overcloud-controller-0** as the **heat-admin** user, run:

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

After logging into a controller, compute, or storage system, you can begin investigating the HA features there.

CHAPTER 4. USING PACEMAKER

In the OpenStack configuration described in Figure 1.1, “OpenStack HA environment deployed through director”, most OpenStack services are running on the three controller nodes. To investigate high availability features of those services, log into any of the controllers as the **heat-admin** user and look at services controlled by Pacemaker.

Output from the Pacemaker **`pcs status`** command includes general Pacemaker information, virtual IP addresses, services, and other Pacemaker information.

For general information about Pacemaker in Red Hat Enterprise Linux, see:

- High Availability Add-On Overview
- High Availability Add-On Administration
- High Availability Add-On Reference

4.1. GENERAL PACEMAKER INFORMATION

The following example shows the general Pacemaker information section of the of the **`pcs status`** command output:

```
$ sudo pcs status
    Cluster name: tripleo_cluster (1)
    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 (2)
    37 resources configured (3)

    Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-
controller-2 ] (4)
    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 ] (5)

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

The main sections of the output show the following information about the cluster:

(1) Name of the cluster.

(2) Number of nodes that are configured for the cluster.

3. Number of resources that are configured for the cluster.
4. Names of the controller nodes that are currently online.
5. Names of the guest nodes that are currently online. Each guest node consists of a complex Bundle Set resource. For more information about bundle sets, see Section 4.3, “OpenStack Services Configured in Pacemaker”.

4.2. VIRTUAL IP ADDRESSES CONFIGURED IN PACEMAKER

Each IPaddr2 resource sets a virtual IP address that clients use to request access to a service. If the Controller Node that is assigned to that IP address fails, the IP address is reassigned to a different controller.

In this example, you can see each controller node that is currently set to listen to a particular virtual IP address.

```
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
```

In the output, each IP address is initially attached to a particular controller. For example, **`192.168.1.150`** is started on **overcloud-controller-0**. However, if that controller fails, the IP address is reassigned to other controllers in the cluster.

The following table describes the IP addresses in the example and shows how each address was originally allocated.

Table 4.1. IP address description and allocation source

IP Address	Description	Allocated From
`192.168.1.150`	Public IP address	**`ExternalAllocationPools`** attribute in the ***network-environment.yaml*** file
`10.200.0.6`	Controller Virtual IP address	Part of the **`dhcp_start`** and **`dhcp_end`** range set to **`10.200.0.5-10.200.0.24`** in the ***undercloud.conf*** file
`172.16.0.10`	Provides access to OpenStack API services on a controller	**`InternalApiAllocationPools`** in the ***network-environment.yaml*** file
`172.18.0.10`	Storage Virtual IP address that provides access to the Glance API and to Swift Proxy services	**`StorageAllocationPools`** attribute in the ***network-environment.yaml*** file

IP Address	Description	Allocated From
172.16.0.11	Provides access to Redis service on a controller	**InternalApiAllocationPools** in the ***network-environment.yaml*** file
172.19.0.10	Provides access to storage management	**StorageMgmtAlloctionPools** in the ***network-environment.yaml*** file

You can view details about a specific IP address that is managed by Pacemaker with the **pcs** command. For example, you can view timeout information or netmask ID.

The following example shows the output of the **pcs** command when you run it on the **ip-192.168.1.150** public IP address.

```
$ 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)
```

If you are logged into the controller that is currently assigned to listen to the IP address 192.168.1.150, you can run the following commands to make sure that the controller is active and that the services are actively listening to that address:

```
$ 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
```

The **ip** command output shows that the *vlan100* interface is listening to both the **192.168.1.150** and **192.168.1.151** IPv4 addresses.

The **netstat** command output shows all processes that are listening to the **192.168.1.150** interface. In addition to the **ntpd** process that is listening at port 123, the **haproxy** process is the only other one listening specifically to **192.168.1.150**.

NOTE

Processes that are listening to all local addresses, such as **0.0.0.0**, are also available through 192.168.1.150. These processes include **sshd**, **mysqld**, **dhclient**, **ntpd**, and so on.

The port numbers in the **netstat** command output can help you to identify the specific service that HAProxy is listening for. You can view the ***/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg*** file to see which services these port numbers represent.

The following list shows several examples of port numbers and the default services that are assigned to them:

- TCP port 6080: **nova_novncproxy**
- TCP port 9696: **neutron**
- TCP port 8000: **heat_cfn**
- TCP port 80: **horizon**
- TCP port 8776: **cinder**

Currently, most services that are defined in the ***haproxy.cfg*** file listen to the **192.168.1.150** IP address on all three controllers. However, only the **controller-0** node is listening externally to the **192.168.1.150** IP address.

Therefore, if the **controller-0** node fails, HAProxy only needs to re-assign 192.168.1.150 to another controller and all other services will already be running on the fallback controller node.

4.3. OPENSTACK SERVICES CONFIGURED IN PACEMAKER

The majority of the services that are managed by the cluster in Red Hat OpenStack Platform 12 and later are configured as **Bundle Set** resources, or *bundles*. These services can be started in the same way on each controller node and are set to always run on each controller.

Bundle

A *bundle* resource handles configuring and replicating the same *container* on all controller nodes, mapping the necessary storage paths to the container directories, and setting specific attributes related to the resource itself.

Container

A container can run different kind of resources, from simple systemd based services like haproxy to complex services like Galera, which requires specific resource agents that controls and set the state of the service on the different nodes.

WARNING

- Using **`docker`** or **`systemctl`** commands to manage bundles or containers is not supported. You can use the commands to check the status of the services, but you should use only Pacemaker to perform actions on these services.
- Docker containers that are controlled by Pacemaker have a **RestartPolicy** set to *no* by Docker. This is to ensure that Pacemaker and not the docker daemon controls the container start and stop actions.

4.3.1. Simple Bundle Set resources (simple bundles)

A simple Bundle Set resource, or *simple bundle*, is a set of containers that each include the same Pacemaker services to be deployed across the controller nodes.

The following example shows the bundle settings from the **`pcs status`** command:

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

For each bundle, you can see the following details:

- The name that Pacemaker assigns to the service
- The reference to the container that is associated with the bundle
- The list of the replicas that are running on the different controllers with their status

4.3.1.1. Simple bundle settings

To see details about a particular bundle service, such as the **haproxy-bundle** service, use the **pcs resource show** command. For example:

```
$ 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)
```

The **haproxy-bundle** example also shows the resource settings for HAProxy. Although HAProxy provides high availability services by load-balancing traffic to selected services, you keep HAProxy itself highly available by configuring it as a Pacemaker bundle service.

From the example output, you see that the bundle configures a Docker container with several specific parameters:

- **image**: Image used by the container, which refers to the local registry of the undercloud.
- **network**: Container network type, which is **"host"** in the example.
- **options**: Specific options for the container.
- **replicas**: Number that indicates how many copies of the container should be created in the cluster. Each bundle includes three containers, one for each controller node.
- **run-command**: System command used to spawn the container.

In addition to the Docker container specification, the bundle configuration contains also the **Storage Mapping** section, in which local path on the host are mapped into the container. Therefore, to check the **haproxy** configuration from the host, you open the ***/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg*** file instead of the ***/etc/haproxy/haproxy.cfg*** file.

4.3.1.2. Checking simple bundle status

You can check the status of the bundle with the **docker** command to launch a command inside the container:

```
$ 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
```

The output shows that the process is running inside the container.

You can also check the bundle status directly from the host:

```
$ 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
```

The discovered **`parent pid`** attribute of the **`haproxy`** command (**`60956`**) can be used to search for the main Docker process that contains the ID of the Docker container (**`39238c5ecb77`**). This is the ID that is shown in the **`docker ps`** command output.

You can run the same commands on any bundle to see the current level of activity and details about the commands that the service runs.

4.3.2. Complex Bundle Set resources (complex bundles)

Complex Bundle Set resources, or *complex bundles*, are Pacemaker services that specify a *resource* configuration in addition to the basic container configuration, which is also included in simple bundles.

This additional configuration is needed to manage *Multi-State* resources, which are services that can have different states depending on which controller node they run.

This example shows a list of complex bundles from the output of the **`pcs status`** command:

```
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
```

In the output, you see that unlike RabbitMQ, the Galera and Redis bundles are run as multi-state resources inside their containers.

For the **galera-bundle** resource, all three controllers are running as Galera masters. For the **redis-bundle** resource, the **overcloud-controller-0** container is running as the master, while the other two controllers are running as slaves.

This means that the Galera service is running under one set of constraints on all three controllers, while the **`redis`** service might run under different constraints on the master and the slave controllers.

The following example shows the output of the **`pcs resource show galera-bundle`** command:

```
[...]
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)
[...]
```

This output shows that unlike in a simple bundle, the **galera-bundle** resource includes explicit resource configuration, which determines all aspects of the multi-state resource.

NOTE

Even though a service might be running on multiple controllers at the same time, the controller itself might not be listening at the IP address that is needed to actually reach those services.

For more information about troubleshooting the Galera resource, see Chapter 6, *Using Galera*.

4.4. PACEMAKER FAILED ACTIONS

If any of the resources fail in any way, they will be listed under the *Failed actions* heading of the **`pcs status`** output. In the following example, the *openstack-cinder-volume* service stopped working on *controller-0*:

```
Failed Actions:
* openstack-cinder-volume_monitor_60000 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
```

In this case, the systemd service *openstack-cinder-volume* needs to be re-enabled. In other cases, you need to track down and fix the problem, then clean up the resources. See Section 7.1, “Correcting Resource Problems on Controllers” for details.

4.5. OTHER PACEMAKER INFORMATION FOR CONTROLLERS

The last sections of the **`pcs status`** output shows information about your power management fencing (IPMI in this case) and the status of the Pacemaker service itself:

```
 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
```

The **my-ipmilan-for-controller** settings show the type of fencing done for each node (**stonith:fence_ipmilan**) and whether or not the IPMI service is stopped or running. The PCSD Status shows that all three controllers are currently online. The Pacemaker service itself consists of three daemons: **corosync**, **pacemaker**, and **pcsd**. Here, all three services are active and enabled.

4.6. FENCING HARDWARE

When a controller node fails a health check, the controller acting as the Pacemaker designated coordinator (DC) uses the Pacemaker **`stonith`** service to fence off the offending node. Stonith is an acronym for the term "Shoot the other node in the head". So, the DC basically kicks the node out of the cluster.

To see how your fencing devices are configured by **`stonith`** for your OpenStack Platform HA cluster, run the following command:

```
$ 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)
```

The **`show --full`** listing shows details about the three controller nodes that relate to fencing. The fence device uses IPMI power management (**fence_ipmilan**) to turn the machines on and off as required. Information about the IPMI interface for each node includes the IP address of the IPMI interface (**10.100.0.51**), the user name to log in as (**admin**) and the password to use (**abc**). You can also see the interval at which each host is monitored (60 seconds).

For more information on fencing with Pacemaker, see "Fencing Configuration" in *Red Hat Enterprise Linux 7 High Availability Add-On Administration*.

CHAPTER 5. USING HAPROXY

HAProxy provides high-availability features to OpenStack by load-balancing traffic to controllers running OpenStack services. The **haproxy** package contains the **haproxy** daemon, which is started from the **systemd** service of the same name, along with logging features and sample configurations. As noted earlier, Pacemaker manages the HAProxy service itself as a highly available service called **haproxy-bundle**.

NOTE

Refer to the KCS solution How can I verify my haproxy.cfg is correctly configured to load balance openstack services? for information on validating an HAProxy configuration.

In Red Hat OpenStack Platform, the director configures multiple OpenStack services to take advantage of the haproxy service. The director does this by configuring those services in the */var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg* file, because HAProxy runs in a dedicated container on each overcloud node.

For each service in that file, you can see the following properties:

- **listen**: The name of the service that is listening for requests
- **bind**: The IP address and TCP port number on which the service is listening
- **server**: The name of each server providing the service, the server's IP address and listening port, and other information.

The *haproxy.cfg* file that is created when you install Red Hat OpenStack Platform with the director identifies 19 different services for HAProxy to manage. The following example shows how the **horizon listen** service is configured in the *haproxy.cfg* file:

```
listen horizon
  bind 10.0.0.106:80 transparent (1)
  bind 172.17.1.12:80 transparent (2)
  mode http (3)
  cookie SERVERID insert indirect nocache
  option forwardfor
  option httpchk (4)
  server controller-0.internalapi.localdomain 172.17.1.21:80 check cookie
controller-2 fall 5 inter 2000 rise 2 (5)
  server controller-1.internalapi.localdomain 172.17.1.23:80 check cookie
controller-2 fall 5 inter 2000 rise 2
  server controller-2.internalapi.localdomain 172.17.1.22:80 check cookie
controller-2 fall 5 inter 2000 rise 2
```

(1) Virtual IP address and port on the External network that provides access to the API network from outside the overcloud.

(2) Virtual IP address and port on the Internal API network to use inside the overcloud.

(3) Type of protocol that the network uses to send and receive information, in this case **http**

(4) Health check to use for network traffic. **httpchk** is the default health check methodology for all API services.

5 Controller node that can accept direct requests from the listed IP addresses. In this case, three controller nodes are available: **controller-0.internalapi.localdomain**, **controller-1.internalapi.localdomain**, and **controller-2.internalapi.localdomain**. The health check is set to run until either two successful attempts (**`rise 2`**) or five unsuccessful attempts (**`fail 5`**).

Here is the list of services managed by HAProxy on the controller nodes:

Table 5.1. Services managed by HAProxy

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

The director also enables *HAProxy Stats* by default on all HA deployments. This feature allows you to view detailed information about data transfer, connections, server states, and the like on the HAProxy Stats page.

The director also sets the IP:Port address through which you can reach the HAProxy Stats page. To find out what this address is, open the ***/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg*** file of any node where HAProxy is installed. The **listen haproxy.stats** section lists this information. For example:

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

In this case, navigate in your browser to *10.200.0.6:1993* and enter the credentials from the **`stats auth`** row to view the HAProxy Stats page.

5.2. REFERENCES

For more information about HAProxy, see HAProxy Configuration (from Load Balancer Administration).

For detailed information about settings you can use in the ***haproxy.cfg*** file, see the documentation in ***/usr/share/doc/haproxy-_VERSION*/configuration.txt_** on any system where the **haproxy** package is installed (such as Controller nodes).

CHAPTER 6. USING GALERA

In a high-availability deployment, Red Hat OpenStack Platform uses the MariaDB Galera Cluster to manage database replication. As described in Section 4.3, “OpenStack Services Configured in Pacemaker”, Pacemaker runs the Galera service using a **bundle set** resource that manages the master/slave status.

You can use the **`pcs status`** command to check if the **`galera-bundle`** service is running, and on which controllers it runs:

```
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. HOSTNAME RESOLUTION

When troubleshooting the MariaDB Galera Cluster, you can verify the hostname resolution. By default, the director binds the Galera resource to a *hostname* rather than to an IP address [1]. Therefore, any problems that prevent hostname resolution, such as misconfigured or failed DNS, might prevent Pacemaker from properly managing the Galera resource.

After you eliminate any hostname resolution problems, you can check the integrity of the cluster itself. To do so, check the write-set replication status on the database of each controller node.

To access MySQL, use the password that was set up by the director during the overcloud deployment. To retrieve the MySQL root password, use the **`hiera`** command:

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

Write-set replication information is stored on each node’s MariaDB database. Each relevant variable uses the prefix ***`wsrep_`***. Therefore, you can query this information directly through the database client:

```
$ 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     |
    +-----------------------+-------+
```

To verify the health and integrity of the MariaDB Galera Cluster, check first whether the cluster is reporting the right number of nodes. Then, check each node if it:

- is part of the correct cluster

- can write to the cluster
- can receive queries and writes from the cluster
- is connected to others within the cluster
- is replicating write-sets to tables in the local database

6.2. DATABASE CLUSTER INTEGRITY

When investigating problems with the MariaDB Galera Cluster, you can check the integrity of the cluster itself. Verifying cluster integrity involves checking specific ***wsrep***_ database variables on each Controller node. To check a database variable, run:

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

Replace ***VARIABLE*** with the ***wsrep***_ database variable you want to check. For example, to view the node's cluster state UUID:

```
$ 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 |
    +--------------------------+--------------------------------------
+
```

The following table lists the different ***wsrep***_ database variables that relate to cluster integrity.

Table 6.1. Database variables to check for cluster integrity

Variable	Summary	Description
wsrep_cluster_state_uuid	Cluster state UUID	The ID of the cluster to which the node belongs. All nodes must have an identical ID. A node with a different ID is not connected to the cluster.
wsrep_cluster_size	Number of nodes in the cluster	You can check this on any single node. If the value is less than the actual number of nodes, then some nodes have either failed or lost connectivity.

Variable	Summary	Description
wsrep_cluster_conf_id	Total number of cluster changes	Determines whether or not the cluster has been split into several components, also known as a *partition*. This is likely caused by a network failure. All nodes must have an identical value. In case some nodes are reporting a different ***wsrep_cluster_conf_id***, check their ***wsrep_cluster_status*** value to see if it can still write to the cluster (**Primary**).
wsrep_cluster_status	Primary component status	Determines whether or not the node can still write to the cluster. If so, then the ***wsrep_cluster_status*** should be **Primary**. Any other value indicates that the node is part of a non-operational partition.

6.3. DATABASE CLUSTER NODE

If you can isolate a Galera cluster problem to a specific node, other ***wsrep_*** database variables can provide clues on the specific problem. You can check these variables in a similar manner as a cluster check, which is described in Section 6.2, “Database Cluster Integrity”.

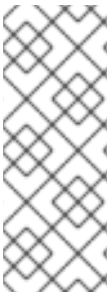
```
$ sudo mysql -B --password="<MYSQL-HIERA-PASSWORD>" -e "SHOW GLOBAL STATUS
LIKE 'VARIABLE';"
```

Replace ***VARIABLE*** with any of the following values:

Table 6.2. Database variables to check for node integrity

Variable	Summary	Description
wsrep_ready	Node ability to accept queries	States whether the node can accept write-sets from the cluster. If so, then ***wsrep_ready*** should be **ON**.

Variable	Summary	Description
wsrep_connected	Node network connectiviry	States whether the node has network connectivity to other nodes. If so, then ***wsrep_connected*** should be **ON**.
wsrep_local_state_comment	Node state	Summarizes the node state. If node can still write to the cluster (ie. if ***wsrep_cluster_status*** is **Primary**, see Section 6.2, "Database Cluster Integrity"), then typical values for ***wsrep_local_state_comment*** are **Joining**, **Waiting on SST**, **Joined**, **Synced**, or **Donor**. If the node is part of a non-operational component, then ***wsrep_local_state_comment*** is set to **Initialized**.



NOTE

A ***wsrep_connected*** variable with an **ON** value might also mean that the node is only connected to some nodes but not all nodes. For example, in cases of a cluster partition, the node might be part of a component that cannot write to the cluster. See Section 6.2, "Database Cluster Integrity" for details.

If ***wsrep_connected*** is **OFF**, then the node is not connected to any cluster components.

6.4. DATABASE REPLICATION PERFORMANCE

If the cluster and its individual nodes are all healthy and stable, you can check replication throughput to benchmark performance. You do this by checking the ***wsrep_*** database variables, similar to Section 6.3, "Database Cluster Node" and Section 6.2, "Database Cluster Integrity".

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

Replace ***VARIABLE*** with any of the following values:

Table 6.3. Database variables to check for cluster performance (replication throughput)

Variable	Summary
wsrep_local_recv_queue_avg	Average size of the local received queue since last query

Variable	Summary
wsrep_local_send_queue_avg	Average send queue length since the last time the variable was queried
wsrep_local_recv_queue_min and ***wsrep_local_recv_queue_max***	The minimum and maximum sizes the local received queue since either variable was last queried
wsrep_flow_control_paused	Fraction of time that the node paused due to *Flow Control* since the last time the variable was queried
wsrep_cert_deps_distance	Average distance between the lowest and highest sequence number (**seqno**) value that can be applied in parallel (such as the potential degree of parallelization)

Each time any of these variables are queried, a **FLUSH STATUS** command resets its value. Benchmarking cluster replication involves querying these values multiple times to see variances. These variances can help you judge how much *Flow Control* is affecting the cluster's performance.

Flow Control is a mechanism used by the cluster to manage replication. When the local received write-set queue exceeds a certain threshold, Flow Control pauses replication in order for the node to catch up. See Flow Control from the Galera Cluster site for more information.

Check the following variables for optimization of different values and benchmarks:

wsrep_local_recv_queue_avg > 0.0

The node cannot apply write-sets as quickly as it receives them, which then triggers *replication throttling*. Check ***wsrep_local_recv_queue_min*** and ***wsrep_local_recv_queue_max*** for a detailed look at this benchmark.

wsrep_local_send_queue_avg > 0.0

As the value of ***wsrep_local_send_queue_avg*** rises, so does the likelihood of replication throttling and network throughput issues. This is especially true as ***wsrep_local_recv_queue_avg*** rises.

wsrep_flow_control_paused > 0.0

Flow Control paused the node. To determine how long the node was paused, multiply the ***wsrep_flow_control_paused*** value with the number of seconds between querying it. For example, if ***wsrep_flow_control_paused = 0.50*** a minute after last checking it, then node replication was paused for 30 seconds. If ***wsrep_flow_control_paused = 1.0***, then the node was paused the entire time since the last query.
Ideally, ***wsrep_flow_control_paused*** should be as close to **0.0** as possible.

wsrep_cert_deps_distance

In case of throttling and pausing, you can check the ***wsrep_cert_deps_distance*** variable to see how many write-sets (on average) can be applied in parallel. Then, check ***wsrep_slave_threads*** to see how many write-sets can actually be applied simultaneously.

wsrep_slave_threads

Configuring a higher ***wsrep_slave_threads*** can help mitigate throttling and pausing. For example, if the ***wsrep_cert_deps_distance*** value is **20**, then doubling ***wsrep_slave_threads*** from **2** to

4 can also double the amount of write-sets that the node can apply. However, ***wsrep_slave_threads*** should be set only as high as the number of CPU cores in the node. If a problematic node already has an optimal ***wsrep_slave_threads*** setting, then consider excluding the node from the cluster as you investigate possible connectivity issues.

---

[1] This method was implemented to allow Galera to start successfully in overclouds that use IPv6 (specifically, to address BZ#1298671).

CHAPTER 7. INVESTIGATING AND FIXING HA CONTROLLER RESOURCES

The **pcs constraint show** command shows any constraints on how services are launched. The output from the command shows constraints related to where each resource is located, the order in which it starts and whether it must be co-located with another resource. If there are any problems, you can try to fix those problems, then clean up the resources.

The following example shows a truncated output from **pcs constraint show** on a controller node:

```
$ 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)
```

This output displays three major constraint types:

Location Constraints

This section shows constraints that are related to where resources are assigned. The first constraint defines a rule that sets the **galera-bundle** resource to be run on nodes with the **galera-role** attribute set to **true**. You can check the attributes of the nodes is by using the **pcs property show** command:

```
$ 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
```

From this output, you can verify that the **`galera-role`** attribute is **`true`** for all the controllers. This means that the **galera-bundle** resource runs only on these nodes. The same concept applies to the other attributes associated with the other location constraints.

The second location constraint relates to the resource ip-192.168.24.15 and specifies that the IP resource runs only on nodes with the **`haproxy-role`** attribute set to **`true`**. This means that the cluster associates the IP address with the **`haproxy`** service, which is necessary to make the services reachable.

The third location constraint shows that the **ipmilan** resource is disabled on each of the controllers.

Ordering Constraints

This section shows the constraint that enforces the virtual IP address resources (IPaddr2) to start before HAProxy. Ordering constraints only apply to IP address resources and HAproxy. All the other resources are managed by systemd, because each service, such as Compute, is expected to be able to support an interruption of a dependent service, such as Galera.

Co-location Constraints

This section shows which resources need to be located together. All virtual IP addresses are linked to the **haproxy-bundle** resource.

7.1. CORRECTING RESOURCE PROBLEMS ON CONTROLLERS

Failed actions relating to the resources managed by the cluster are listed by the **`pcs status`** command. There are many different kinds of problems that can occur. In general, you can approach problems in the following ways:

Controller problem

If health checks to a controller are failing, log into the controller and check if services can start up without problems. Service startup problems could indicate a communication problem between controllers. Other indications of communication problems between controllers include:

- A controller gets fenced disproportionately more than other controllers, and/or
- A suspiciously large amount of services are failing from a specific controller.

Individual resource problem

If services from a controller are generally working, but an individual resource is failing, see if you can figure out the problem from the **pcs status** messages. If you need more information, log into the controller where the resource is failing and try some of the steps below.
Apart from IPs and core bundle resources (Galera, Rabbit and Redis) the only A/P resource managed by the cluster is **openstack-cinder-volume**. If this resource has an associated failed action, a good approach is to check the status from a **systemctl** perspective. So, once you have identified the node on which the resource is failing (for example **overcloud-controller-0**), it is possible to check the status of the resource:

```
[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.
```

After you correct the failed resource, you can run the **`pcs resource cleanup`** command to reset the status and the fail count of the resource. Then, after finding and fixing a problem with the **`openstack-cinder-volume`** resource, run:

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

CHAPTER 8. INVESTIGATING HA CEPH NODES

When deployed with Ceph storage, Red Hat OpenStack Platform uses **ceph-mon** as a monitor daemon for the Ceph cluster. The director deploys this daemon on all controller nodes.

To check whether the Ceph Monitoring service is running, use:

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

On the controllers, as well as on the Ceph Nodes, you can see how Ceph is configured by viewing the ***/etc/ceph/ceph.conf*** file. For example:

```
[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
```

Here, all three controller nodes (**overcloud-controller-0**, **-1**, and **-2**) are set to monitor the Ceph cluster (**mon_initial_members**). The 172.19.0.11/24 network (VLAN 203) is used as the Storage Management Network and provides a communications path between the controller and Ceph Storage Nodes.

The three Ceph Storage Nodes are on a separate network. As you can see, the IP addresses for those three nodes are on the Storage Network (VLAN 202) and are defined as 172.18.0.15, 172.18.0.16, and 172.18.0.17.

To check the current status of a Ceph node, log into that node and run the following command:

```
# 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
```

From the **ceph -s** output, you can see that the health of the Ceph cluster is OK (**HEALTH_OK**). There are three Ceph monitor services running on the three **overcloud-controller** nodes. Also shown here are the IP addresses and ports each is listening on.

For more information about Red Hat Ceph, see the Red Hat Ceph product page.

APPENDIX A. BUILDING THE RED HAT OPENSTACK PLATFORM 12 HA ENVIRONMENT

The Deploying an Overcloud with Containerized Red Hat Ceph guide provides instructions for deploying the type of highly available OpenStack environment described in this document. The Director Installation and Usage guide was also used for reference throughout the process.

A.1. HARDWARE SPECIFICATION

The following tables show the specifications used by the deployment tested for this document. For better results, increase the CPU, memory, storage, or NICs on your own test deployment.

Table A.1. Physical Computers

Number of Computers	Assigned as…	CPUs	Memory	Disk space	Power mgmt.	NICs
1	Director node	4	6144 MB	40 GB	IPMI	2 (1 external; 1 on Provisioning) + 1 IPMI
3	Controller nodes	4	6144 MB	40 GB	IPMI	3 (2 bonded on Overcloud; 1 on Provisioning) + 1 IPMI
3	Ceph Storage nodes	4	6144 MB	40 GB	IPMI	3 (2 bonded on Overcloud; 1 on Provisioning) + 1 IPMI
2	Compute node (add more as needed)	4	6144 MB	40 GB	IPMI	3 (2 bonded on Overcloud; 1 on Provisioning) + 1 IPMI

The following list describes the general functions and connections associated with each non-director assignment:

Controller nodes

Most OpenStack services, other than storage, run on these controller nodes. All services are replicated across the three nodes (some active-active; some active-passive). Three nodes are required for reliable HA.

Ceph storage nodes

Storage services run on these nodes, providing pools of Ceph storage areas to the compute nodes. Again, three nodes are needed for HA.

Compute nodes

Virtual machines actually run on these compute nodes. You can have as many compute nodes as you need to meet your capacity requirements, including the ability to shut down compute nodes and migrate virtual machines between those nodes. Compute nodes must be connected to the storage network (so the VMs can access storage) and Tenant network (so VMs can access VMs on other compute nodes and also access public networks, to make their services available).

Table A.2. Physical and Virtual Networks

Physical NICs	Reason for Network	VLANs	Used to…
eth0	Provisioning network (undercloud)	N/A	Manage all nodes from director (undercloud)
eth1 and eth2	Controller/External (overcloud)	N/A	Bonded NICs with VLANs
	External Network	VLAN 100	Allow access from outside world to Tenant networks, Internal API, and OpenStack Horizon Dashboard
	Internal API	VLAN 201	Provide access to the internal API between compute and controller nodes
	Storage access	VLAN 202	Connect compute nodes to underlying Storage media
	Storage management	VLAN 203	Manage storage media
	Tenant network	VLAN 204	Provide tenant network services to OpenStack

The following are also required:

Provisioning network switch

This switch must be able to connect the director system (undercloud) to all computers in the Red Hat OpenStack Platform environment (overcloud). The NIC on each overcloud node that is connected to this switch must be able to PXE boot from the director. Also check that the switch has portfast set to enabled.

Controller/External network switch

This switch must be configured to do VLAN tagging for the VLANs shown in Figure 1. Only VLAN 100 traffic should be allowed to external networks.

Fencing Hardware

Hardware defined for use with Pacemaker is supported in this configuration. Supported fencing devices can be determined using the **stonith** Pacemaker tool. See Fencing the Controller Nodes for more information.

A.2. UNDERCLOUD CONFIGURATION FILES

This section shows relevant configuration files from the test configuration used for this document. If you change IP address ranges, consider making a diagram similar to Figure 1.1, “OpenStack HA environment deployed through director” to track your resulting address settings.

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. OVERCLOUD CONFIGURATION FILES

The following configuration files reflect the actual overcloud settings from the deployment used for this document.

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

This file identifies the services that HAProxy manages. It contains the settings that define the services monitored by HAProxy. This file exists and is the same on all Controller nodes.

```
# 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-sslv3
  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 (Controller Nodes)

This file defines the cluster infrastructure, and is available on all Controller nodes.

```
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 Nodes)

This file contains Ceph high availability settings, including the hostnames and IP addresses of monitoring hosts.

```
[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
```