Red Hat OpenStack Platform 17.1

Managing high availability services

Plan, deploy, and manage high availability in Red Hat OpenStack Platform

Last Updated: 2024-05-23

Red Hat OpenStack Platform 17.1 Managing high availability services

Plan, deploy, and manage 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, use the Red Hat OpenStack Platform director to create configurations that offer high availability and load-balancing across all major services in OpenStack.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 4

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 5

CHAPTER 1. RED HAT OPENSTACK PLATFORM HIGH AVAILABILITY OVERVIEW AND PLANNING 6
1.1. RED HAT OPENSTACK PLATFORM HIGH AVAILABILITY SERVICES 6
1.1.1. Service types 6
1.1.2. Service modes 6
1.2. PLANNING HIGH AVAILABILITY HARDWARE ASSIGNMENTS 7
1.3. PLANNING HIGH AVAILABILITY NETWORKING 7
1.4. ACCESSING THE HIGH AVAILABILITY ENVIRONMENT 8
1.5. ADDITIONAL RESOURCES 8

CHAPTER 2. EXAMPLE DEPLOYMENT: HIGH AVAILABILITY CLUSTER WITH COMPUTE AND CEPH 9
2.1. EXAMPLE HIGH AVAILABILITY HARDWARE SPECIFICATIONS 9
2.2. EXAMPLE HIGH AVAILABILITY NETWORK SPECIFICATIONS 10
2.3. EXAMPLE HIGH AVAILABILITY UNDERCLOUD CONFIGURATION FILES 11
2.4. EXAMPLE HIGH AVAILABILITY OVERCLOUD CONFIGURATION FILES 14
2.5. ADDITIONAL RESOURCES 20

CHAPTER 3. MANAGING HIGH AVAILABILITY SERVICES WITH PACEMAKER 21
3.1. PACEMAKER RESOURCE BUNDLES AND CONTAINERS 21
3.1.1. Simple Bundle Set resources (simple bundles) 21
3.1.2. Complex Bundle Set resources (complex bundles) 22
3.2. CHECKING PACEMAKER CLUSTER STATUS 24
3.3. CHECKING BUNDLE STATUS IN A HIGH AVAILABILITY CLUSTER 25
3.4. VIEWING RESOURCE INFORMATION FOR VIRTUAL IPS IN A HIGH AVAILABILITY CLUSTER 26
3.5. VIEWING NETWORK INFORMATION FOR VIRTUAL IPS IN A HIGH AVAILABILITY CLUSTER 27
3.6. CHECKING FENCING AGENT AND PACEMAKER DAEMON STATUS 29
3.7. ADDITIONAL RESOURCES 30

CHAPTER 4. FENCING CONTROLLER NODES WITH STONITH 31
4.1. SUPPORTED FENCING AGENTS 31
4.2. DEPLOYING FENCING ON THE OVERCLOUD 32
4.3. TESTING FENCING ON THE OVERCLOUD 35
4.4. VIEWING STONITH DEVICE INFORMATION 36
4.5. FENCING PARAMETERS 37
4.6. ADDITIONAL RESOURCES 38

CHAPTER 5. LOAD BALANCING TRAFFIC WITH HAPROXY 39
5.1. HOW HAPROXY WORKS 39
5.2. VIEWING HAPROXY STATS 40
5.3. ADDITIONAL RESOURCES 40

CHAPTER 6. CONFIGURING THE OVERCLOUD TO USE AN EXTERNAL LOAD BALANCER 42
6.1. PREPARING YOUR ENVIRONMENT FOR AN EXTERNAL LOAD BALANCER 42
6.2. CONFIGURING THE OVERCLOUD NETWORK FOR AN EXTERNAL LOAD BALANCER 44
6.3. CREATING AN EXTERNAL LOAD BALANCER ENVIRONMENT FILE 45
6.4. CONFIGURING SSL FOR EXTERNAL LOAD BALANCING 47
6.5. DEPLOYING THE OVERCLOUD WITH AN EXTERNAL LOAD BALANCER 48
6.6. ADDITIONAL RESOURCES 49

CHAPTER 7. EXAMPLE CONFIGURATION: OVERCLOUD WITH AN EXTERNAL HAPROXY LOAD BALANCER 50

7.1. EXAMPLE HAPROXY CONFIGURATION FILE 50
7.1.1. Global configuration parameters: Example HAProxy configuration file 54
7.1.2. Default values configuration parameters: Example HAProxy configuration file 55
7.1.3. Service-level configuration parameters: Example HAProxy configuration file 55
7.2. CONFIGURATION PARAMETERS FOR SERVICES THAT USE LOAD BALANCING 56

CHAPTER 8. MANAGING DATABASE REPLICATION WITH GALERA 68
8.1. VERIFYING HOSTNAME RESOLUTION IN A MARIADB CLUSTER 68
8.2. CHECKING MARIADB CLUSTER INTEGRITY 69
8.3. CHECKING DATABASE NODE INTEGRITY IN A MARIADB CLUSTER 70
8.4. TESTING DATABASE REPLICATION PERFORMANCE IN A MARIADB CLUSTER 71
8.5. ADDITIONAL RESOURCES 74

CHAPTER 9. TROUBLESHOOTING HIGH AVAILABILITY RESOURCES 75
9.1. VIEWING RESOURCE CONSTRAINTS IN A HIGH AVAILABILITY CLUSTER 75
9.2. INVESTIGATING PACEMAKER RESOURCE PROBLEMS 77
9.3. INVESTIGATING SYSTEMD RESOURCE PROBLEMS 78

CHAPTER 10. MONITORING A HIGH AVAILABILITY RED HAT CEPH STORAGE CLUSTER 80
10.1. CHECKING RED HAT CEPH MONITORING SERVICE STATUS 80
10.2. CHECKING RED HAT CEPH MONITORING CONFIGURATION 80
10.3. CHECKING RED HAT CEPH NODE STATUS 81
10.4. ADDITIONAL RESOURCES 81

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: Create Issue
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

CHAPTER 1. RED HAT OPENSTACK PLATFORM HIGH AVAILABILITY OVERVIEW AND PLANNING

Red Hat OpenStack Platform (RHOSP) high availability (HA) is a collection of services that orchestrate failover and recovery for your deployment. When you plan your HA deployment, ensure that you review the considerations for different aspects of the environment, such as hardware assignments and network configuration.

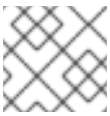
1.1. RED HAT OPENSTACK PLATFORM HIGH AVAILABILITY SERVICES

Red Hat OpenStack Platform (RHOSP) employs several technologies to provide the services required to implement high availability (HA). These services include Galera, RabbitMQ, Redis, HAProxy, individual services that Pacemaker manages, and Systemd and plain container services that Podman manages.

1.1.1. Service types

Core container

Core container services are 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. You use Pacemaker to launch, manage, and troubleshoot core container services.



NOTE

RHOSP uses the MariaDB Galera Cluster to manage database replication.

Active-passive

Active-passive services run on one Controller node at a time, and include services such as **openstack-cinder-volume**. To move an active-passive service, you must use Pacemaker to ensure that the correct stop-start sequence is followed.

Systemd and plain container

Systemd and plain container services are independent services that can withstand a service interruption. Therefore, if you restart a high availability service such as Galera, you do not need to manually restart any other service, such as **nova-api**. You can use systemd or Podman to directly manage systemd and plain container services.

When orchestrating your HA deployment, director uses templates and Puppet modules to ensure that all services are configured and launched correctly. In addition, when troubleshooting HA issues, you must interact with services in the HA framework using the **podman** command or the **systemctl** command.

1.1.2. Service modes

HA services can run in one of the following modes:

Active-active

Pacemaker runs the same service on multiple Controller nodes, and uses HAProxy to distribute traffic across the nodes or to a specific Controller with a single IP address. In some cases, HAProxy distributes traffic to active-active services with Round Robin scheduling. You can add more Controller nodes to improve performance.

IMPORTANT

Active-active mode is supported only in distributed compute node (DCN) architecture at Edge sites.

Active-passive

Services that are unable to run in active-active mode must run in active-passive mode. In this mode, only one instance of the service is active at a time. For example, HAProxy uses stick-table options to direct incoming Galera database connection requests to a single back-end service. This helps prevent too many simultaneous connections to the same data from multiple Galera nodes.

1.2. PLANNING HIGH AVAILABILITY HARDWARE ASSIGNMENTS

When you plan hardware assignments, consider the number of nodes that you want to run in your deployment, as well as the number of Virtual Machine (vm) instances that you plan to run on Compute nodes.

Controller nodes

Most non-storage services run on Controller nodes. All services are replicated across the three nodes and are configured as active-active or active-passive services. A high availability (HA) environment requires a minimum of three nodes.

Red Hat Ceph Storage nodes

Storage services run on these nodes and provide pools of Red Hat Ceph Storage areas to the Compute nodes. A minimum of three nodes are required.

Compute nodes

Virtual machine (VM) instances run on Compute nodes. You can deploy as many Compute nodes as you need to meet your capacity requirements, as well as migration and reboot operations. You must connect Compute nodes to the storage network and to the project network to ensure that VMs can access storage nodes, VMs on other Compute nodes, and public networks.

STONITH

You must configure a STONITH device for each node that is a part of the Pacemaker cluster in a highly available overcloud. Deploying a highly available overcloud without STONITH is not supported. For more information on STONITH and Pacemaker, see Fencing in a Red Hat High Availability Cluster and Support Policies for RHEL High Availability Clusters .

1.3. PLANNING HIGH AVAILABILITY NETWORKING

When you plan the virtual and physical networks, consider the provisioning network switch configuration and the external network switch configuration.

In addition to the network configuration, you must deploy the following components:

Provisioning network switch

- This switch must be able to connect the undercloud to all the physical computers in the overcloud.
- The NIC on each overcloud node that is connected to this switch must be able to PXE boot from the undercloud.
- The **portfast** parameter must be enabled.

Controller/External network switch

- This switch must be configured to perform VLAN tagging for the other VLANs in the deployment.
- Allow only VLAN 100 traffic to external networks.

Networking hardware and keystone endpoint

- To prevent a Controller node network card or network switch failure disrupting overcloud services availability, ensure that the keystone admin endpoint is located on a network that uses bonded network cards or networking hardware redundancy.
  If you move the keystone endpoint to a different network, such as **internal_api**, ensure that the undercloud can reach the VLAN or subnet. For more information, see the Red Hat Knowledgebase solution How to migrate Keystone Admin Endpoint to internal_api network .

1.4. ACCESSING THE HIGH AVAILABILITY ENVIRONMENT

To investigate high availability (HA) nodes, use the **stack** user to log in to the overcloud nodes and run the **openstack server list** command to view the status and details of the nodes.

Prerequisites

- High availability is deployed and running.

Procedure

1. In a running HA environment, log in to the undercloud as the **stack** user.
2. Identify the IP addresses of your overcloud nodes:

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

3. Log in to one of the overcloud nodes:

```
(undercloud) $ ssh tripleo-admin@<node_IP>
```

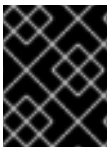
   Replace **<node_ip>** with the IP address of the node that you want to log in to.

1.5. ADDITIONAL RESOURCES

- Chapter 2, *Example deployment: High availability cluster with Compute and Ceph*

CHAPTER 2. EXAMPLE DEPLOYMENT: HIGH AVAILABILITY CLUSTER WITH COMPUTE AND CEPH

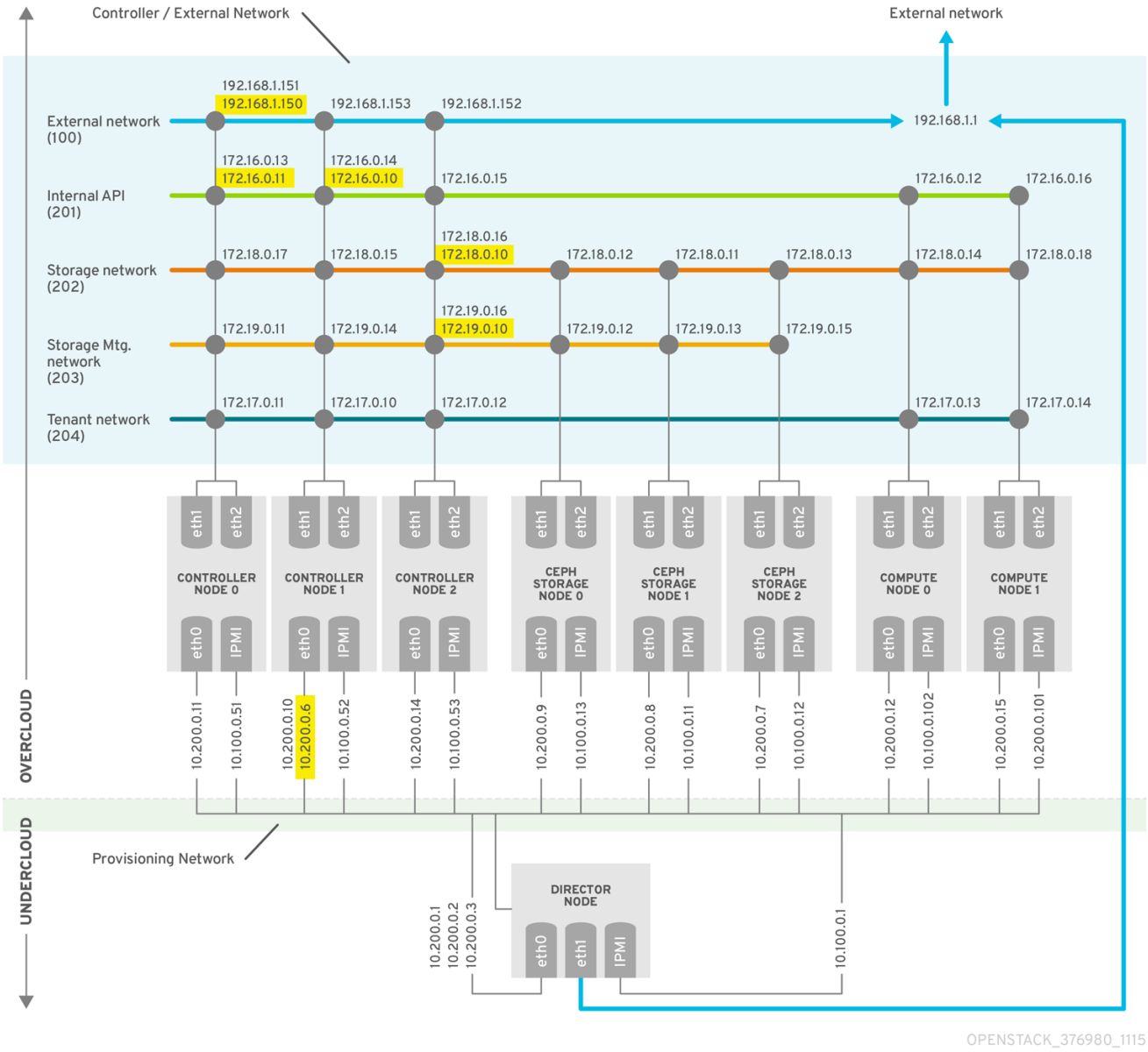
This example scenario shows the architecture, hardware and network specifications, and the undercloud and overcloud configuration files for a high availability deployment with the OpenStack Compute service and Red Hat Ceph Storage.



IMPORTANT

This deployment is intended to use as a reference for test environments and is not supported for production environments.

Figure 2.1. Example high availability deployment architecture



2.1. EXAMPLE HIGH AVAILABILITY HARDWARE SPECIFICATIONS

The example HA deployment uses a specific hardware configuration. You can adjust the CPU, memory, storage, or NICs as needed in your own test deployment.

Table 2.1. Physical computers

Number of Computers	Purpose	CPUs	Memory	Disk Space	Power Management	NICs
1	undercloud node	4	24 GB	40 GB	IPMI	2 (1 external; 1 on provisioning) + 1 IPMI
3	Controller nodes	4	24 GB	40 GB	IPMI	3 (2 bonded on overcloud; 1 on provisioning) + 1 IPMI
3	Ceph Storage nodes	4	24 GB	40 GB	IPMI	3 (2 bonded on overcloud; 1 on provisioning) + 1 IPMI
2	Compute nodes (add more as needed)	4	24 GB	40 GB	IPMI	3 (2 bonded on overcloud; 1 on provisioning) + 1 IPMI

2.2. EXAMPLE HIGH AVAILABILITY NETWORK SPECIFICATIONS

The example HA deployment uses a specific virtual and physical network configuration. You can adjust the configuration as needed in your own test deployment.

NOTE

This example does not include hardware redundancy for the control plane and the provisioning network where the overcloud keystone admin endpoint is configured. For information about planning your high availability networking, see Section 1.3, “Planning high availability networking”.

Table 2.2. Physical and virtual networks

Physical NICs	Purpose	VLANs	Description
eth0	Provisioning network (undercloud)	N/A	Manages all nodes from director (undercloud)

Physical NICs	Purpose	VLANs	Description
eth1 and eth2	Controller/External (overcloud)	N/A	Bonded NICs with VLANs
	External network	VLAN 100	Allows access from outside the environment to the project networks, internal API, and OpenStack Horizon Dashboard
	Internal API	VLAN 201	Provides access to the internal API between Compute nodes and Controller nodes
	Storage access	VLAN 202	Connects Compute nodes to storage media
	Storage management	VLAN 203	Manages storage media
	Project network	VLAN 204	Provides project network services to RHOSP

2.3. EXAMPLE HIGH AVAILABILITY UNDERCLOUD CONFIGURATION FILES

The example HA deployment uses the undercloud configuration files **instackenv.json**, **undercloud.conf**, and **network-environment.yaml**.

instackenv.json

```
{
    "nodes": [
      {
        "pm_password": "testpass",
        "memory": "24",
        "pm_addr": "10.100.0.11",
        "mac": [
          "2c:c2:60:3b:b3:94"
        ],
        "pm_type": "ipmi",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
      },
      {
```

```
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.12",
    "mac": [
      "2c:c2:60:51:b7:fb"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.13",
    "mac": [
      "2c:c2:60:76:ce:a5"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.51",
    "mac": [
      "2c:c2:60:08:b1:e2"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.52",
    "mac": [
      "2c:c2:60:20:a1:9e"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.53",
    "mac": [
```

```
            "2c:c2:60:58:10:33"
          ],
          "pm_type": "ipmi",
          "disk": "40",
          "arch": "x86_64",
          "cpu": "1",
          "pm_user": "admin"
        },
        {
          "pm_password": "testpass",
          "memory": "24",
          "pm_addr": "10.100.0.101",
          "mac": [
            "2c:c2:60:31:a9:55"
          ],
          "pm_type": "ipmi",
          "disk": "40",
          "arch": "x86_64",
          "cpu": "2",
          "pm_user": "admin"
        },
        {
          "pm_password": "testpass",
          "memory": "24",
          "pm_addr": "10.100.0.102",
          "mac": [
            "2c:c2:60:0d:e7:d1"
          ],
          "pm_type": "ipmi",
          "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"
```

2.4. EXAMPLE HIGH AVAILABILITY OVERCLOUD CONFIGURATION FILES

The example HA deployment uses the overcloud configuration files **haproxy.cfg**, **corosync.cfg**, and **ceph.cfg**.

/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg (Controller nodes)

This file identifies the services that HAProxy manages. It contains the settings for the services that HAProxy monitors. This file is identical 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:!IDEA:!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 the monitoring hosts.

```
[global]
osd_pool_default_pgp_num = 128
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
```

2.5. ADDITIONAL RESOURCES

- Deploying Red Hat Ceph Storage and Red Hat OpenStack Platform together with director
- Chapter 1, *Red Hat OpenStack Platform high availability overview and planning*

CHAPTER 3. MANAGING HIGH AVAILABILITY SERVICES WITH PACEMAKER

The Pacemaker service manages core container and active-passive services, such as Galera, RabbitMQ, Redis, and HAProxy. You use Pacemaker to view and manage general information about the managed services, virtual IP addresses, power management, and fencing.

3.1. PACEMAKER RESOURCE BUNDLES AND CONTAINERS

Pacemaker manages Red Hat OpenStack Platform (RHOSP) services as Bundle Set resources, or bundles. Most of these services are active-active services that start in the same way and always run on each Controller node.

Pacemaker manages the following resource types:

Bundle

A bundle resource configures and replicates the same container on all Controller nodes, maps the necessary storage paths to the container directories, and sets specific attributes related to the resource itself.

Container

A container can run different kinds of resources, from simple **systemd** services like HAProxy to complex services like Galera, which requires specific resource agents that control and set the state of the service on the different nodes.

IMPORTANT

- You cannot use **podman** or **systemctl** to manage bundles or containers. You can use the commands to check the status of the services, but you must use Pacemaker to perform actions on these services.
- Podman containers that Pacemaker controls have a **RestartPolicy** set to **no** by Podman. This is to ensure that Pacemaker, and not Podman, controls the container start and stop actions.

3.1.1. 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 that you want to deploy across the Controller nodes.

The following example shows a list of simple bundles from the output of the **pcs status** command:

```
Podman container set: haproxy-bundle [192.168.24.1:8787/rhosp-rhel8/openstack-
haproxy:pcmklatest]
  haproxy-bundle-podman-0    (ocf::heartbeat:podman):    Started overcloud-controller-0
  haproxy-bundle-podman-1    (ocf::heartbeat:podman):    Started overcloud-controller-1
  haproxy-bundle-podman-2    (ocf::heartbeat:podman):    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 and status of replicas that are running on the different Controller nodes.

The following example shows the settings for the **haproxy-bundle** simple bundle:

```
$ sudo pcs resource show haproxy-bundle
Bundle: haproxy-bundle
 Podman: image=192.168.24.1:8787/rhosp-rhel8/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 example shows the following information about the containers in the bundle:

- **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**: Indicates how many copies of the container must run in the cluster. Each bundle includes three containers, one for each Controller node.
- **run-command**: System command used to spawn the container.
- **Storage Mapping**: Mapping of the local path on each host to the container. The **haproxy** configuration is located in the **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** file instead of the **/etc/haproxy/haproxy.cfg** file.

NOTE

Although HAProxy provides high availability services by load balancing traffic to selected services, you configure HAProxy as a highly available service by managing it as a Pacemaker bundle service.

3.1.2. 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 that is included in simple bundles.

This configuration is needed to manage multi-state resources, which are services that can have different states depending on the Controller node they run on.

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

```
Podman container set: rabbitmq-bundle [192.168.24.1:8787/rhosp-rhel8/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
Podman container set: galera-bundle [192.168.24.1:8787/rhosp-rhel8/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
Podman container set: redis-bundle [192.168.24.1:8787/rhosp-rhel8/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
```

This output shows the following information about each complex bundle:

- RabbitMQ: All three Controller nodes run a standalone instance of the service, similar to a simple bundle.
- Galera: All three Controller nodes are running as Galera masters under the same constraints.
- Redis: The **overcloud-controller-0** container is running as the master, while the other two Controller nodes are running as slaves. Each container type might run under different constraints.

The following example shows the settings for the **galera-bundle** complex bundle:

```
[...]
Bundle: galera-bundle
 Podman: image=192.168.24.1:8787/rhosp-rhel8/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 that determines all aspects of the multi-state resource.

NOTE

Although a service can run on multiple Controller nodes at the same time, the Controller node itself might not be listening at the IP address that is required to reach those services. For information about how to check the IP address of a service, see Section 3.4, “Viewing resource information for virtual IPs in a high availability cluster”.

3.2. CHECKING PACEMAKER CLUSTER STATUS

You can check the status of the Pacemaker cluster in any node where Pacemaker is running, and view information about the number of resources that are active and running.

Prerequisites

- High availability is deployed and running.

Procedure

1. Log in to any Controller node as the **tripleo-admin** user.

```
$ ssh tripleo-admin@overcloud-controller-0
```

2. Run the **pcs status** command:

```
[tripleo-admin@overcloud-controller-0 ~]  $ sudo pcs status
```

Example output:

```
Cluster name: tripleo_cluster
Stack: corosync
Current DC: overcloud-controller-1 (version 2.0.1-4.el8-0eb7991564) - 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
37 resources configured

Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-controller-2 ]
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 ]
```

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

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

- **Cluster name**: Name of the cluster.
- **[NUM] nodes configured**: Number of nodes that are configured for the cluster.
- **[NUM] resources configured**: Number of resources that are configured for the cluster.
- **Online**: Names of the Controller nodes that are currently online.
- **GuestOnline**: 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 3.1, "Pacemaker resource bundles and containers".

3.3. CHECKING BUNDLE STATUS IN A HIGH AVAILABILITY CLUSTER

You can check the status of a bundle from an undercloud node or log in to one of the Controller nodes to check the bundle status directly.

Prerequisites

- High availability is deployed and running.

Procedure

Use one of the following options:

- Log in to an undercloud node and check the bundle status, in this example **haproxy-bundle**:

```
$ sudo podman exec -it haproxy-bundle-podman-0 ps -efww | grep haproxy*
```

Example output:

```
root           7       1  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
haproxy       11       7  0 06:08 ?        00:00:17 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -
Ws
```

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

- Log in to a Controller node and check the bundle status, in this example **haproxy**:

```
$ ps -ef | grep haproxy*
```

Example output:

```
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
-Ws
42454      17819   17774  0 06:08 ?        00:00:21 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ws
root      288508  237714  0 07:04 pts/0    00:00:00 grep --color=auto haproxy*
[root@controller-0 ~]# ps -ef | grep -e 17774 -e 17819
```

```
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
-Ws
42454      17819   17774  0 06:08 ?        00:00:22 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ws
root      301950  237714  0 07:07 pts/0    00:00:00 grep --color=auto -e 17774 -e 17819
```

3.4. VIEWING RESOURCE INFORMATION FOR VIRTUAL IPS IN A HIGH AVAILABILITY CLUSTER

To check the status of all virtual IPs (VIPs) or a specific VIP, run the **pcs resource show** command with the relevant options. Each IPaddr2 resource sets a virtual IP address that clients use to request access to a service. If the Controller node with that IP address fails, the IPaddr2 resource reassigns the IP address to a different Controller node.

Prerequisites

- High availability is deployed and running.

Procedure

1. Log in to any Controller node as the **tripleo-admin** user.

```
$ ssh tripleo-admin@overcloud-controller-0
```

2. Use one of the following options:

   - Show all resources that use virtual IPs by running the **pcs resource show** command with the **--full** option:

```
$ sudo pcs resource show --full
```

     Example output:

```
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
```

     Each IP address is initially attached to a specific Controller node. For example, **192.168.1.150** is started on **overcloud-controller-0**. However, if that Controller node fails, the IP address is reassigned to other Controller nodes in the cluster.

     The following table describes the IP addresses in the example output and shows the original allocation of each IP address.

Table 3.1. IP address description and allocation source

IP Address	Description	Allocated From

IP Address	Description	Allocated From
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
192.168.1.150	Public IP address	**ExternalAllocationPools** attribute in the **network-environment.yaml** file
172.16.0.10	Provides access to OpenStack API services on a Controller node	**InternalApiAllocationPools** in the **network-environment.yaml** file
172.16.0.11	Provides access to Redis service on a Controller node	**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
172.19.0.10	Provides access to storage management	**StorageMgmtAlloctionPools** in the **network-environment.yaml** file

- View a specific VIP address by running the **pcs resource show** command with the name of the resource that uses that VIP, in this example **ip-192.168.1.150**:

```
$ sudo pcs resource show ip-192.168.1.150
```

Example output:

```
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)
```

3.5. VIEWING NETWORK INFORMATION FOR VIRTUAL IPS IN A HIGH AVAILABILITY CLUSTER

You can view the network interface information for a Controller node that is assigned to a specific virtual IP (VIP), and view port number assignments for a specific service.

Prerequisites

- High availability is deployed and running.

Procedure

1. Log in to the Controller node that is assigned to the IP address you want to view and run the **ip addr show** command on the network interface, in this example **vlan100**:

```
$ ip addr show vlan100
```

Example output:

```
 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
```

2. Run the **netstat** command to show all processes that listen to the IP address, in this example **192.168.1.150.haproxy**:

```
$ sudo netstat -tupln | grep "192.168.1.150.haproxy"
```

Example output:

```
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
```

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**.

3. View the default port number assignments and the services they listen to by opening the configuration file for the HA service, in this example **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg**:

- TCP port 6080: **nova_novncproxy**
- TCP port 9696: **neutron**
- TCP port 8000: **heat_cfn**

- TCP port 80: **horizon**
- TCP port 8776: **cinder**
  In this example, most services that are defined in the **haproxy.cfg** file listen to the **192.168.1.150** IP address on all three Controller nodes. 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 node and all other services will already be running on the fallback Controller node.

3.6. CHECKING FENCING AGENT AND PACEMAKER DAEMON STATUS

You can check the status of the fencing agent and the status of the Pacemaker daemons in any node where Pacemaker is running, and view information about the number of Controller nodes that are active and running.

Prerequisites

- High availability is deployed and running.

Procedure

1. Log in to any Controller node as the **tripleo-admin** user.

```
$ ssh tripleo-admin@overcloud-controller-0
```

2. Run the **pcs status** command:

```
[tripleo-admin@overcloud-controller-0 ~]  $ sudo pcs status
```

Example output:

```
 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 output shows the following sections of the **pcs status** command output:

- **my-ipmilan-for-controller**: Shows the type of fencing for each Controller node (**stonith:fence_ipmilan**) and whether or not the IPMI service is stopped or running.
- **PCSD Status**: Shows that all three Controller nodes are currently online.

- **Daemon Status**: Shows the status of the three Pacemaker daemons: **corosync**, **pacemaker**, and **pcsd**. In the example, all three services are active and enabled.

3.7. ADDITIONAL RESOURCES

- Configuring and Managing High Availability Clusters

CHAPTER 4. FENCING CONTROLLER NODES WITH STONITH

Fencing is the process of isolating a failed node to protect the cluster and the cluster resources. Without fencing, a failed node might result in data corruption in a cluster. Director uses Pacemaker to provide a highly available cluster of Controller nodes.

Pacemaker uses a process called STONITH to fence failed nodes. STONITH is an acronym for "Shoot the other node in the head". STONITH is disabled by default and requires manual configuration so that Pacemaker can control the power management of each node in the cluster.

If a Controller node fails a health check, the Controller node that acts as the Pacemaker designated coordinator (DC) uses the Pacemaker **stonith** service to fence the impacted Controller node.

IMPORTANT

Deploying a highly available overcloud without STONITH is not supported. You must configure a STONITH device for each node that is a part of the Pacemaker cluster in a highly available overcloud. For more information on STONITH and Pacemaker, see Fencing in a Red Hat High Availability Cluster and Support Policies for RHEL High Availability Clusters.

4.1. SUPPORTED FENCING AGENTS

When you deploy a high availability environment with fencing, you can choose the fencing agents based on your environment needs. To change the fencing agent, you must configure additional parameters in the **fencing.yaml** file.

Red Hat OpenStack Platform (RHOSP) supports the following fencing agents:

Intelligent Platform Management Interface (IPMI)

Default fencing mechanism that Red Hat OpenStack Platform (RHOSP) uses to manage fencing.

STONITH Block Device (SBD)

The SBD (Storage-Based Death) daemon integrates with Pacemaker and a watchdog device to arrange for nodes to reliably shut down when fencing is triggered and in cases where traditional fencing mechanisms are not available.

IMPORTANT

- SBD fencing is not supported in clusters with remote bare metal or virtual machine nodes that use **pacemaker_remote**, so it is not supported if your deployment uses Instance HA.
- **fence_sbd** and **sbd poison-pill** fencing with block storage devices are not supported.
- SBD fencing is only supported with compatible watchdog devices. For more information, see Support Policies for RHEL High Availability Clusters - sbd and fence_sbd.

fence_kdump

Use in deployments with the **kdump** crash recovery service. If you choose this agent, ensure that you have enough disk space to store the dump files.

You can configure this agent as a secondary mechanism in addition to the IPMI, **fence_rhevm**, or Redfish fencing agents. If you configure multiple fencing agents, make sure that you allocate enough time for the first agent to complete the task before the second agent starts the next task.

IMPORTANT

- RHOSP director supports only the configuration of the **fence_kdump** STONITH agent, and not the configuration of the full **kdump** service that the fencing agent depends on. For information about configuring the **kdump** service, see the article How do I configure fence_kdump in a Red Hat Pacemaker cluster.
- **fence_kdump** is not supported if the Pacemaker network traffic interface uses the **ovs_bridges** or **ovs_bonds** network device. To enable **fence_kdump**, you must change the network device to **linux_bond** or **linux_bridge**.

Redfish

Use in deployments with servers that support the DMTF Redfish APIs. To specify this agent, change the value of the **agent** parameter to **fence_redfish** in the **fencing.yaml** file. For more information about Redfish, see the DTMF Documentation.

Multi-layered fencing

You can configure multiple fencing agents to support complex fencing use cases. For example, you can configure IPMI fencing together with **fence_kdump**. The order of the fencing agents determines the order in which Pacemaker triggers each mechanism.

Additional resources

- Section 4.2, "Deploying fencing on the overcloud"
- Section 4.3, "Testing fencing on the overcloud"
- Section 4.5, "Fencing parameters"

4.2. DEPLOYING FENCING ON THE OVERCLOUD

To deploy fencing on the overcloud, first review the state of STONITH and Pacemaker and configure the **fencing.yaml** file. Then, deploy the overcloud and configure additional parameters. Finally, test that fencing is deployed correctly on the overcloud.

Prerequisites

- Choose the correct fencing agent for your deployment. For the list of supported fencing agents, see Section 4.1, "Supported fencing agents" .
- Ensure that you can access the **nodes.json** file that you created when you registered your nodes in director. This file is a required input for the **fencing.yaml** file that you generate during deployment.
- The **nodes.json** file must contain the MAC address of one of the network interfaces (NICs) on the node. For more information, see Registering Nodes for the Overcloud.

Procedure

1. Log in to each Controller node as the **tripleo-admin** user.
2. Verify that the cluster is running:

```
$ sudo pcs status
```

Example output:

```
Cluster name: openstackHA
Last updated: Wed Jun 24 12:40:27 2015
Last change: Wed Jun 24 11:36:18 2015
Stack: corosync
Current DC: lb-c1a2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
141 Resources configured
```

3. Verify that STONITH is disabled:

```
$ sudo pcs property show
```

Example output:

```
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

4. Depending on the fencing agent that you want to use, choose one of the following options:
   - If you use the IPMI or RHV fencing agent, generate the **fencing.yaml** environment file:

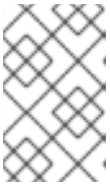
```
(undercloud) $ openstack overcloud generate fencing --output fencing.yaml nodes.json
```

NOTE

This command converts **ilo** and **drac** power management details to IPMI equivalents.

   - If you use a different fencing agent, such as STONITH Block Device (SBD), **fence_kdump**, or Redfish, or if you use pre-provisioned nodes, create the **fencing.yaml** file manually.
5. SBD fencing only: Add the following parameter to the **fencing.yaml** file:

```
parameter_defaults:
  ExtraConfig:
    pacemaker::corosync::enable_sbd: true
```



NOTE

This step is applicable to initial overcloud deployments only. For more information about how to enable SBD fencing on an existing overcloud, see Enabling sbd fencing in RHEL 7 and 8.

6. Multi-layered fencing only: Add the level-specific parameters to the generated **fencing.yaml** file:

```
parameter_defaults:
  EnableFencing: true
  FencingConfig:
    devices:
      level1:
      - agent: [VALUE]
        host_mac: aa:bb:cc:dd:ee:ff
        params:
          <parameter>: <value>
      level2:
      - agent: fence_agent2
        host_mac: aa:bb:cc:dd:ee:ff
        params:
          <parameter>: <value>
```

Replace **<parameter>** and **<value>** with the actual parameters and values that the fencing agent requires.

7. Run the **overcloud deploy** command and include the **fencing.yaml** file and any other environment files that are relevant for your deployment:

```
openstack overcloud deploy --templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e ~/templates/network-environment.yaml \
-e ~/templates/storage-environment.yaml --ntp-server pool.ntp.org --neutron-network-type
vxlan --neutron-tunnel-types vxlan \
-e fencing.yaml
```

8. SBD fencing only: Set the watchdog timer device interval and check that the interval is set correctly.

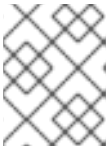
```
# pcs property set stonith-watchdog-timeout=<interval>
# pcs property show
```

Verification

1. Log in to the overcloud as the **heat-admin** user and ensure that Pacemaker is configured as the resource manager:

```
$ source stackrc
$ openstack server list | grep controller
$ ssh tripleo-admin@<controller-x_ip>
$ sudo pcs status | grep fence
stonith-overcloud-controller-x (stonith:fence_ipmilan): Started overcloud-controller-y
```

In this example, Pacemaker is configured to use a STONITH resource for each of the Controller nodes that are specified in the **fencing.yaml** file.



NOTE

You must not configure the **fence-resource** process on the same node that it controls.

2. Check the fencing resource attributes. The STONITH attribute values must match the values in the **fencing.yaml** file:

```
$ sudo pcs stonith show <stonith-resource-controller-x>
```

Additional Resources

- Section 4.3, "Testing fencing on the overcloud"
- Section 4.5, "Fencing parameters"
- Exploring RHEL High Availability's Components - sbd and fence_sbd

4.3. TESTING FENCING ON THE OVERCLOUD

To test that fencing works correctly, trigger fencing by closing all ports on a Controller node and restarting the server.

IMPORTANT

This procedure deliberately drops all connections to the Controller node, which causes the node to restart.

Prerequisites

- Fencing is deployed and running on the overcloud. For information on how to deploy fencing, see Section 4.2, "Deploying fencing on the overcloud" .
- Controller node is available for a restart.

Procedure

1. Log in to a Controller node as the **stack** user and source the credentials file:

```
$ source stackrc
$ openstack server list | grep controller
$ ssh tripleo-admin@<controller-x_ip>
```

2. Change to the **root** user and close all connections to the Controller node:

```
$ sudo -i
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 5016 -j ACCEPT &&
iptables -A INPUT -p udp -m state --state NEW -m udp --dport 5016 -j ACCEPT &&
```

```
iptables -A INPUT ! -i lo -j REJECT --reject-with icmp-host-prohibited &&
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT &&
iptables -A OUTPUT -p tcp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT -p udp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT ! -o lo -j REJECT --reject-with icmp-host-prohibited
```

3. From a different Controller node, locate the fencing event in the Pacemaker log file:

```
$ ssh tripleo-admin@<controller-x_ip>
$ less /var/log/cluster/corosync.log
(less): /fenc*
```

If the STONITH service performed the fencing action on the Controller, the log file shows a fencing event.

4. Wait a few minutes and then verify that the restarted Controller node is running in the cluster again by running the **pcs status** command. If you can see the Controller node that you restarted in the output, fencing functions correctly.

4.4. VIEWING STONITH DEVICE INFORMATION

To see how STONITH configures your fencing devices, run the **pcs stonith show --full** command from the overcloud.

Prerequisites

- Fencing is deployed and running on the overcloud. For information on how to deploy fencing, see Section 4.2, “Deploying fencing on the overcloud” .

Procedure

- Show the list of Controller nodes and the status of their STONITH devices:

```
$ 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)
```

This output shows the following information for each resource:

  - IPMI power management service that the fencing device uses to turn the machines on and off as needed, such as **fence_ipmilan**.
  - IP address of the IPMI interface, such as **10.100.0.51**.
  - User name to log in with, such as **admin**.

- Password to use to log in to the node, such as **abc**.
- Interval in seconds at which each host is monitored, such as **60s**.

4.5. FENCING PARAMETERS

When you deploy fencing on the overcloud, you generate the **fencing.yaml** file with the required parameters to configure fencing.

The following example shows the structure of the **fencing.yaml** environment file:

```
parameter_defaults:
  EnableFencing: true
  FencingConfig:
    devices:
    - agent: fence_ipmilan
      host_mac: 11:11:11:11:11:11
      params:
        ipaddr: 10.0.0.101
        lanplus: true
        login: admin
        passwd: InsertComplexPasswordHere
        pcmk_host_list: host04
        privlvl: administrator
```

This file contains the following parameters:

EnableFencing

Enables the fencing functionality for Pacemaker-managed nodes.

FencingConfig

Lists the fencing devices and the parameters for each device:

- **agent**: Fencing agent name.
- **host_mac**: The mac address in lowercase of the provisioning interface or any other network interface on the server. You can use this as a unique identifier for the fencing device.

IMPORTANT

Do not use the MAC address of the IPMI interface.

- **params**: List of fencing device parameters.

Fencing device parameters

Lists the fencing device parameters. This example shows the parameters for the IPMI fencing agent:

- **auth**: IPMI authentication type (**md5**, **password**, or none).
- **ipaddr**: IPMI IP address.
- **ipport**: IPMI port.
- **login**: Username for the IPMI device.

- **passwd**: Password for the IPMI device.
- **lanplus**: Use lanplus to improve security of connection.
- **privlvl**: Privilege level on IPMI device
- **pcmk_host_list**: List of Pacemaker hosts.

Additional resources

- Section 4.2, “Deploying fencing on the overcloud”
- Section 4.1, “Supported fencing agents”

4.6. ADDITIONAL RESOURCES

- "Configuring fencing in a Red Hat High Availability cluster"

CHAPTER 5. LOAD BALANCING TRAFFIC WITH HAPROXY

The HAProxy service provides load balancing of traffic to Controller nodes in the high availability cluster, as well as logging and sample configurations. The **haproxy** package contains the **haproxy** daemon, which corresponds to the **systemd** service of the same name. Pacemaker manages the HAProxy service as a highly available service called **haproxy-bundle**.

5.1. HOW HAPROXY WORKS

Director can configure most Red Hat OpenStack Platform services to use the HAProxy service. Director configures those services in the **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** file, which instructs HAProxy to run in a dedicated container on each overcloud node.

The following table shows the list of services that HAProxy manages:

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

For each service in the **haproxy.cfg** 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 Controller node server that uses HAProxy, the IP address and listening port, and additional information about the server.

The following example shows the OpenStack Block Storage (cinder) service configuration in the **haproxy.cfg** file:

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

This example output shows the following information about the OpenStack Block Storage (cinder) service:

- **172.16.0.10:8776**: Virtual IP address and port on the Internal API network (VLAN201) to use within the overcloud.
- **192.168.1.150:8776**: Virtual IP address and port on the External network (VLAN100) that provides access to the API network from outside the overcloud.
- **8776**: Port number on which the OpenStack Block Storage (cinder) service is listening.
- **server**: Controller node names and IP addresses. HAProxy can direct requests made to those IP addresses to one of the Controller nodes listed in the **server** output.
- **httpchk**: Enables health checks on the Controller node servers.
- **fall 5**: Number of failed health checks to determine that the service is offline.
- **inter 2000**: Interval between two consecutive health checks in milliseconds.
- **rise 2**: Number of successful health checks to determine that the service is running.

For more information about settings you can use in the **haproxy.cfg** file, see the **/usr/share/doc/haproxy-[VERSION]/configuration.txt** file on any node where the **haproxy** package is installed.

5.2. VIEWING HAPROXY STATS

By default, the director also enables HAProxy Stats, or statistics, on all HA deployments. With this feature, you can view detailed information about data transfer, connections, and server states on the HAProxy Stats page.

The director also sets the **IP:Port** address that you use to reach the HAProxy Stats page and stores the information in the **haproxy.cfg** file.

Prerequisites

- High availability is deployed and running.

Procedure

1. Open the **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** file in any Controller node where HAProxy is installed.
2. Locate the **listen haproxy.stats** section:

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

3. In a Web browser, navigate to **10.200.0.6:1993** and enter the credentials from the **stats auth** row to view the HAProxy Stats page.

5.3. ADDITIONAL RESOURCES

- HAProxy 1.8 documentation
- How can I verify my haproxy.cfg is correctly configured to load balance openstack services?

CHAPTER 6. CONFIGURING THE OVERCLOUD TO USE AN EXTERNAL LOAD BALANCER

In Red Hat OpenStack Platform (RHOSP), the overcloud uses multiple Controller nodes together as a high availability cluster to ensure maximum operational performance for your OpenStack services. The cluster also provides load balancing for OpenStack services, which evenly distributes traffic to the Controller nodes and reduces server overload for each node.

By default, the overcloud uses an open source tool called HAProxy to manage load balancing. HAProxy load balances traffic to the Controller nodes that run OpenStack services. The **haproxy** package contains the **haproxy** daemon that listens to incoming traffic, and includes logging features and sample configurations.

The overcloud also uses the high availability resource manager Pacemaker to control HAProxy as a highly available service. This means that HAProxy runs on each Controller node and distributes traffic according to a set of rules that you define in each configuration.

You can also use an external load balancer to perform this distribution. For example, your organization might use a dedicated hardware-based load balancer to handle traffic distribution to the Controller nodes. To define the configuration for an external load balancer and the overcloud creation, you perform the following processes:

1. Install and configure an external load balancer.
2. Configure and deploy the overcloud with heat template parameters to integrate the overcloud with the external load balancer. This requires the IP addresses of the load balancer and of the potential nodes.

Before you configure your overcloud to use an external load balancer, ensure that you deploy and run high availability on the overcloud.

6.1. PREPARING YOUR ENVIRONMENT FOR AN EXTERNAL LOAD BALANCER

To prepare your environment for an external load balancer, first create a node definition template and register blank nodes with director. Then, inspect the hardware of all nodes and manually tag nodes into profiles.

Use the following workflow to prepare your environment:

- Create a node definition template and register blank nodes with Red Hat OpenStack Platform director. A node definition template **instackenv.json** is a JSON format file and contains the hardware and power management details to register nodes.
- Inspect the hardware of all nodes. This ensures that all nodes are in a manageable state.
- Manually tag nodes into profiles. These profile tags match the nodes to flavors. The flavors are then assigned to a deployment role.

Procedure

1. Log in to the director host as the **stack** user and source the director credentials:

```
$ source ~/stackrc
```

2. Create a node definition template **instackenv.json** and copy and edit the following example based on your environment:

```
{
    "nodes":[
        {
            "mac":[
                "bb:bb:bb:bb:bb:bb"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"ipmi",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.205"
        },
        {
            "mac":[
                "cc:cc:cc:cc:cc:cc"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"ipmi",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.206"
        },
        {
            "mac":[
                "dd:dd:dd:dd:dd:dd"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"ipmi",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.207"
        },
        {
            "mac":[
                "ee:ee:ee:ee:ee:ee"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"ipmi",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.208"
```

```
        }
    ]
}
```

3. Save the file to the home directory of the **stack** user, **/home/stack/instackenv.json**, then import it into director and register the nodes to the director:

```
$ openstack overcloud node import ~/instackenv.json
```

4. Assign the kernel and ramdisk images to all nodes:

```
$ openstack overcloud node configure
```

5. Inspect the hardware attributes of each node:

```
$ openstack overcloud node introspect --all-manageable
```

IMPORTANT

The nodes must be in the manageable state. Ensure that this process runs to completion. This process usually takes 15 minutes for bare metal nodes.

6. Get the list of your nodes to identify their UUIDs:

```
$ openstack baremetal node list
```

7. Manually tag each node to a specific profile by adding a profile option in the **properties/capabilities** parameter for each node. For example, to tag three nodes to use a Controller profile and one node to use a Compute profile, use the following commands:

```
$ openstack baremetal node set 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 --property
capabilities='profile:control,boot_option:local'
$ openstack baremetal node set 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a --property
capabilities='profile:control,boot_option:local'
$ openstack baremetal node set 5e3b2f50-fcd9-4404-b0a2-59d79924b38e --property
capabilities='profile:control,boot_option:local'
$ openstack baremetal node set 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13 --property
capabilities='profile:compute,boot_option:local'
```

The **profile:compute** and **profile:control** options tag the nodes into each respective profile.

Additional resources

- Planning your overcloud

6.2. CONFIGURING THE OVERCLOUD NETWORK FOR AN EXTERNAL LOAD BALANCER

To configure the network for the overcloud, isolate your services to use specific network traffic and then configure the network environment file for your local environment. This file is a heat environment file that describes the overcloud network environment, points to the network interface configuration templates, and defines the subnets and VLANs for your network and the IP address ranges.

Procedure

1. To configure the node interfaces for each role, customize the following network interface templates:
   - To configure a single NIC with VLANs for each role, use the example templates in the following directory:

     ```
     /usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans
     ```

   - To configure bonded NICs for each role, use the example templates in the following directory:

     ```
     /usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans
     ```

2. Create a network environment file by copying the file from **/home/stack/network-environment.yaml** and editing the content based on your environment.

Additional resources

- Basic network isolation
- Custom composable networks
- Custom network interface templates
- Overcloud networks

6.3. CREATING AN EXTERNAL LOAD BALANCER ENVIRONMENT FILE

To deploy an overcloud with an external load balancer, create a new environment file with the required configuration. In this example file, several virtual IPs are configured on the external load balancer, one virtual IP on each isolated network, and one for the Redis service, before the overcloud deployment starts. Some of the virtual IPs can be identical if the overcloud node NICs configuration supports the configuration.

Procedure

- Use the following example environment file **external-lb.yaml** to create the environment file, and edit the content based on your environment.

```
parameter_defaults:
  ControlFixedIPs: [{'ip_address':'192.0.2.250'}]
  PublicVirtualFixedIPs: [{'ip_address':'172.16.23.250'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.20.250'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.16.21.250'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.16.19.250'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.16.20.249'}]
  # IPs assignments for the Overcloud Controller nodes. Ensure these IPs are from each
respective allocation pools defined in the network environment file.
  ControllerIPs:
    external:
    - 172.16.23.150
    - 172.16.23.151
    - 172.16.23.152
```

```
    internal_api:
    - 172.16.20.150
    - 172.16.20.151
    - 172.16.20.152
    storage:
    - 172.16.21.150
    - 172.16.21.151
    - 172.16.21.152
    storage_mgmt:
    - 172.16.19.150
    - 172.16.19.151
    - 172.16.19.152
    tenant:
    - 172.16.22.150
    - 172.16.22.151
    - 172.16.22.152
    # CIDRs
    external_cidr: "24"
    internal_api_cidr: "24"
    storage_cidr: "24"
    storage_mgmt_cidr: "24"
    tenant_cidr: "24"
  RedisPassword: p@55w0rd!
  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    AodhApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: internal_api
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage_mgmt
    SwiftProxyNetwork: storage
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage_mgmt
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

NOTE

- The **parameter_defaults** section contains the VIP and IP assignments for each network. These settings must match the same IP configuration for each service on the load balancer.
- The **parameter_defaults** section defines an administrative password for the Redis service (RedisPassword) and contains the **ServiceNetMap** parameter, which maps each OpenStack service to a specific network. The load balancing configuration requires this services remap.

6.4. CONFIGURING SSL FOR EXTERNAL LOAD BALANCING

To configure encrypted endpoints for the external load balancer, create additional environment files that enable SSL to access endpoints and then install a copy of your SSL certificate and key on your external load balancing server. By default, the overcloud uses unencrypted endpoints services.

Prerequisites

- If you are using an IP address or domain name to access the public endpoints, choose one of the following environment files to include in your overcloud deployment:
  - To access the public endpoints with a domain name service (DNS), use the file **/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-dns.yaml**.
  - To access the public endpoints with an IP address, use the file **/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-ip.yaml**.

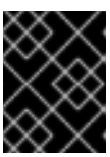
Procedure

1. If you use a self-signed certificate or if the certificate signer is not in the default trust store on the overcloud image, inject the certificate into the overcloud image by copying the **inject-trust-anchor.yaml** environment file from the heat template collection:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/inject-trust-anchor.yaml
~/templates/
```

2. Open the file in a text editor and copy the contents of the root certificate authority file to the **SSLRootCertificate** parameter:

```
parameter_defaults:
  SSLRootCertificate: |
  -----BEGIN CERTIFICATE-----
  MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
  ...
  sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
  -----END CERTIFICATE-----
```



IMPORTANT

The certificate authority content requires the same indentation level for all new lines.

3. Change the resource URL for the **OS::TripleO::NodeTLSCAData:** parameter to an absolute URL:

```
resource_registry:
  OS::TripleO::NodeTLSCAData: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/tls/ca-inject.yaml
```

4. Optional: If you use a DNS hostname to access the overcloud through SSL/TLS, create a new environment file **~/templates/cloudname.yaml** and define the hostname of the overcloud endpoints:

```
parameter_defaults:
  CloudName: overcloud.example.com
```

   Replace **overcloud.example.com** with the DNS hostname for the overcloud endpoints.

6.5. DEPLOYING THE OVERCLOUD WITH AN EXTERNAL LOAD BALANCER

To deploy an overcloud that uses an external load balancer, run the **openstack overcloud deploy** and include the additional environment files and configuration files for the external load balancer.

Prerequisites

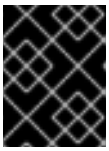
- Environment is prepared for an external load balancer. For more information about how to prepare your environment, see Section 6.1, “Preparing your environment for an external load balancer”
- Overcloud network is configured for an external load balancer. For information about how to configure your network, see Section 6.2, “Configuring the overcloud network for an external load balancer”
- External load balancer environment file is prepared. For information about how to create the environment file, see Section 6.3, “Creating an external load balancer environment file”
- SSL is configured for external load balancing. For information about how to configure SSL for external load balancing, see Section 6.4, “Configuring SSL for external load balancing”

Procedure

1. Deploy the overcloud with all the environment and configuration files for an external load balancer:

```
$ openstack overcloud deploy --templates /
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml /
  -e ~/network-environment.yaml /
  -e /usr/share/openstack-tripleo-heat-templates/environments/external-loadbalancer-vip.yaml
/
  -e ~/external-lb.yaml --control-scale 3 --compute-scale 1 --control-flavor control --compute-
flavor compute /
  -e <SSL/TLS endpoint environment file> /
  -e <DNS hostname environment file> /
  -e <root certificate injection environment file> /
  -e <additional_options_if_needed>
```

Replace the values in angle brackets **<>** with the file paths you defined for your environment.



IMPORTANT

You must add the network environment files to the command in the order listed in this example.

This command includes the following environment files:

- **network-isolation.yaml**: Network isolation configuration file.
- **network-environment.yaml**: Network configuration file.
- **external-loadbalancer-vip.yaml**: External load balancing virtual IP addresses configuration file.
- **external-lb.yaml**: External load balancer configuration file. You can also set the following options for this file and adjust the values for your environment:
  - **--control-scale 3**: Scale the Controller nodes to three.
  - **--compute-scale 3**: Scale the Compute nodes to three.
  - **--control-flavor control**: Use a specific flavor for the Controller nodes.
  - **--compute-flavor compute**: Use a specific flavor for the Compute nodes.
- SSL/TLS environment files:
  - **SSL/TLS endpoint environment file**: Environment file that defines how to connect to public endpoinst. Use **tls-endpoints-public-dns.yaml** or **tls-endpoints-public-ip.yaml**.
  - (Optional) **DNS hostname environment file**: The environment file to set the DNS hostname.
  - **Root certificate injection environment file**: The environment file to inject the root certificate authority.

During the overcloud deployment process, Red Hat OpenStack Platform director provisions your nodes. This process takes some time to complete.

2. To view the status of the overcloud deployment, enter the following commands:

```
$ source ~/stackrc
$ openstack stack list --nested
```

6.6. ADDITIONAL RESOURCES

- Load balancing traffic with HAProxy .

CHAPTER 7. EXAMPLE CONFIGURATION: OVERCLOUD WITH AN EXTERNAL HAPROXY LOAD BALANCER

This example configuration shows an overcloud that uses a federated HAProxy server to provide external load balancing. You can choose a different external load balancer based on your environment requirements.

The example configuration includes the following elements:

- An external load balancing server that runs HAProxy.
- One Red Hat OpenStack Platform (RHOSP) director node.
- An overcloud that consists of 3 Controller nodes in a highly available cluster and 1 Compute node.
- Network isolation with VLANs.

The example uses the following IP address assignments for each network:

- Internal API: **172.16.20.0/24**
- Tenant: **172.16.22.0/24**
- Storage: **172.16.21.0/24**
- Storage management: **172.16.19.0/24**
- External: **172.16.23.0/24**

These IP ranges include IP assignments for the Controller nodes and virtual IPs that the load balancer binds to OpenStack services.

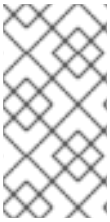
7.1. EXAMPLE HAPROXY CONFIGURATION FILE

The example file shows the internal HAProxy configuration parameters. You can use the sample configuration parameters as a basis for configuring your external load balancer.

The HAProxy configuration file contains the following sections:

- Global configuration
- Defaults configuration
- Services configurations

Director provides this configuration in the **/etc/haproxy/haproxy.conf** file on each Controller node for non-containerized environments, and in the **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** file for containerized environments.



NOTE

In addition to the global, default, and services parameters, you must also configure other HAProxy parameters. For more information about HAProxy parameters, see the *HAProxy Configuration Manual* located in **/usr/share/doc/haproxy-*/configuration.txt** on the Controller nodes or on any system where the **haproxy** package is installed.

Example HAProxy configuration file

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

defaults
  log  global
  mode  tcp
  retries  3
  timeout  http-request 10s
  timeout  queue 1m
  timeout  connect 10s
  timeout  client 1m
  timeout  server 1m
  timeout  check 10s

listen aodh
  bind 172.16.20.250:8042
  bind 172.16.20.250:8042
  mode http
  server overcloud-controller-0 172.16.20.150:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.252:8042 check fall 5 inter 2000 rise 2

listen ceilometer
  bind 172.16.20.250:8777
  bind 172.16.23.250:8777
  server overcloud-controller-0 172.16.20.150:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8777 check fall 5 inter 2000 rise 2

listen cinder
  bind 172.16.20.250:8776
  bind 172.16.23.250:8776
  server overcloud-controller-0 172.16.20.150:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8776 check fall 5 inter 2000 rise 2

listen glance_api
  bind 172.16.23.250:9292
  bind 172.16.21.250:9292
  server overcloud-controller-0 172.16.21.150:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.21.151:9292 check fall 5 inter 2000 rise 2
```

```
  server overcloud-controller-2 172.16.21.152:9292 check fall 5 inter 2000 rise 2

listen glance_registry
  bind 172.16.20.250:9191
  server overcloud-controller-0 172.16.20.150:9191 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:9191 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:9191 check fall 5 inter 2000 rise 2

listen gnocchi
  bind 172.16.23.250:8041
  bind 172.16.21.250:8041
  mode http
  server overcloud-controller-0 172.16.20.150:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8041 check fall 5 inter 2000 rise 2

listen heat_api
  bind 172.16.20.250:8004
  bind 172.16.23.250:8004
  mode http
  server overcloud-controller-0 172.16.20.150:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8004 check fall 5 inter 2000 rise 2

listen heat_cfn
  bind 172.16.20.250:8000
  bind 172.16.23.250:8000
  server overcloud-controller-0 172.16.20.150:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.152:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.151:8000 check fall 5 inter 2000 rise 2

listen heat_cloudwatch
  bind 172.16.20.250:8003
  bind 172.16.23.250:8003
  server overcloud-controller-0 172.16.20.150:8003 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8003 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8003 check fall 5 inter 2000 rise 2

listen horizon
  bind 172.16.20.250:80
  bind 172.16.23.250:80
  mode http
  cookie SERVERID insert indirect nocache
  server overcloud-controller-0 172.16.20.150:80 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:80 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:80 check fall 5 inter 2000 rise 2

listen keystone_admin
  bind 172.16.23.250:35357
  bind 172.16.20.250:35357
  server overcloud-controller-0 172.16.20.150:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:35357 check fall 5 inter 2000 rise 2

listen keystone_admin_ssh
  bind 172.16.20.250:22
```

```
  server overcloud-controller-0 172.16.20.150:22 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:22 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:22 check fall 5 inter 2000 rise 2

listen keystone_public
  bind 172.16.20.250:5000
  bind 172.16.23.250:5000
  server overcloud-controller-0 172.16.20.150:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:5000 check fall 5 inter 2000 rise 2

listen mysql
  bind 172.16.20.250:3306
  option tcpka
  option httpchk
  stick on dst
  stick-table type ip size 1000
  timeout client 0
  timeout server 0
  server overcloud-controller-0 172.16.20.150:3306 backup check fall 5 inter 2000 on-marked-down
shutdown-sessions port 9200 rise 2
  server overcloud-controller-1 172.16.20.151:3306 backup check fall 5 inter 2000 on-marked-down
shutdown-sessions port 9200 rise 2
  server overcloud-controller-2 172.16.20.152:3306 backup check fall 5 inter 2000 on-marked-down
shutdown-sessions port 9200 rise 2

listen neutron
  bind 172.16.20.250:9696
  bind 172.16.23.250:9696
  server overcloud-controller-0 172.16.20.150:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:9696 check fall 5 inter 2000 rise 2

listen nova_ec2
  bind 172.16.20.250:8773
  bind 172.16.23.250:8773
  server overcloud-controller-0 172.16.20.150:8773 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8773 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8773 check fall 5 inter 2000 rise 2

listen nova_metadata
  bind 172.16.20.250:8775
  server overcloud-controller-0 172.16.20.150:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8775 check fall 5 inter 2000 rise 2

listen nova_novncproxy
  bind 172.16.20.250:6080
  bind 172.16.23.250:6080
  balance source
  server overcloud-controller-0 172.16.20.150:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:6080 check fall 5 inter 2000 rise 2

listen nova_osapi
  bind 172.16.20.250:8774
```

```
  bind 172.16.23.250:8774
  server overcloud-controller-0 172.16.20.150:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8774 check fall 5 inter 2000 rise 2

listen nova_placement
  bind 172.16.20.250:8778
  bind 172.16.23.250:8778
  mode http
  server overcloud-controller-0 172.16.20.150:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8778 check fall 5 inter 2000 rise 2

listen panko
  bind 172.16.20.250:8779 transparent
  bind 172.16.23.250:8779 transparent
  server overcloud-controller-0 172.16.20.150:8779 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8779 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8779 check fall 5 inter 2000 rise 2

listen redis
  bind 172.16.20.249:6379
  balance first
  option tcp-check
  tcp-check send AUTH\ p@55w0rd!\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-0 172.16.20.150:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:6379 check fall 5 inter 2000 rise 2

listen swift_proxy_server
  bind 172.16.23.250:8080
  bind 172.16.21.250:8080
  server overcloud-controller-0 172.16.21.150:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.21.151:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.21.152:8080 check fall 5 inter 2000 rise 2
```

7.1.1. Global configuration parameters: Example HAProxy configuration file

The global configuration parameters section defines a set of process-wide parameters for the load balancer. You can use the example parameters from the configuration file to configure your external load balancer. Adjust the parameter values based on your environment.

Global configuration parameters

```
global
  daemon
  user  haproxy
  group  haproxy
```

```
  log  /dev/log local0
  maxconn  10000
  pidfile  /var/run/haproxy.pid
```

The example shows the following parameters:

- **daemon**: Run as a background process.
- **user haproxy** and **group haproxy**: Define the Linux user and group that owns the process.
- **log**: Defines the syslog server to use.
- **maxconn**: Sets the maximum number of concurrent connections to the process.
- **pidfile**: Sets the file to use for the process IDs.

7.1.2. Default values configuration parameters: Example HAProxy configuration file

The default values configuration parameters section defines a set of default values to use when running the external load balancer services. You can use the example parameters from the configuration file to configure your external load balancer. Adjust the parameter values based on your environment.

Default values configuration parameters

```
defaults
  log  global
  mode  tcp
  retries  3
  timeout  http-request 10s
  timeout  queue 1m
  timeout  connect 10s
  timeout  client 1m
  timeout  server 1m
  timeout  check 10s
```

The example shows the following parameters:

- **log**: Enables logging for the service. The **global** value means that the logging functions use the **log** parameters from the **global** section.
- **mode**: Defines the protocol to use. In this case, the default is TCP.
- **retries**: Sets the number of retries to perform on a server before reporting a connection failure.
- **timeout**: Sets the maximum time to wait for a particular function. For example, **timeout http-request** sets the maximum time to wait for a complete HTTP request.

7.1.3. Service-level configuration parameters: Example HAProxy configuration file

The service-level configuration parameters section defines a set of parameters to use when load balancing traffic to a specific Red Hat OpenStack Platform (RHOSP) service. You can use the example parameters from the configuration file to configure your external load balancer. Adjust the parameter values based on your environment, and copy the section for each service that you want to load balance.

Service-level configuration parameters

This example shows the configuration parameters for the **ceilometer** service.

```
listen ceilometer
  bind 172.16.20.250:8777
  bind 172.16.23.250:8777
  server overcloud-controller-0 172.16.20.150:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8777 check fall 5 inter 2000 rise 2
```

Each service that you want to load balance must correspond to a section in the configuration file. Each service configuration includes the following parameters:

- **listen**: The name of the service that listens for requests.
- **bind**: The IP address and TCP port number the on which the service listens. Each service binds a different address that represents a different network traffic type.
- **server**: The name of each server that provides the service, the server IP address and listening port, and connection parameters:
- **check**: (Optional) Enables health checks.
- **fall 5**: (Optional) After five failed health checks, the service is considered offline.
- **inter 2000**: (Optional) The interval between two consecutive health checks set to 2000 milliseconds, or 2 seconds.
- **rise 2**: (Optional) After two successful health checks, the service is considered operational.

In the **ceilometer** example, the service identifies the IP addresses and ports on which the ceilometer service is offered as **172.16.20.2500:8777** and **172.16.23.250:8777**. HAProxy directs the requests for those addresses to **overcloud-controller-0** (172.16.20.150:8777), **overcloud-controller-1** (172.16.20.151:8777), or **overcloud-controller-2** (172.16.0.152:8777).

Additional resources

- Section 7.2, “Configuration parameters for services that use load balancing”

7.2. CONFIGURATION PARAMETERS FOR SERVICES THAT USE LOAD BALANCING

For each service in the overcloud that uses load balancing, use the following examples as a guide to configure your external load balancer. Adjust the parameter values based on your environment, and copy the section for each service that you want to load balance.

NOTE

Most services use the default health check configuration:

- The interval between two consecutive health checks set to 2000 milliseconds, or 2 seconds.
- After two successful health checks, a server is considered operational.
- After five failed health checks, the service is considered offline.

Each service indicates the default health check or additional options in the **Other information** section of that service.

aodh

Port number: 8042

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen aodh
  bind 172.16.20.250:8042
  bind 172.16.23.250:8042
  server overcloud-controller-0 172.16.20.150:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8042 check fall 5 inter 2000 rise 2
```

ceilometer

Port number: 8777

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen ceilometer
  bind 172.16.20.250:8777
  bind 172.16.23.250:8777
  server overcloud-controller-0 172.16.20.150:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8777 check fall 5 inter 2000 rise 2
```

-

cinder

Port number: 8776

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen cinder
  bind 172.16.20.250:8776
  bind 172.16.23.250:8776
  server overcloud-controller-0 172.16.20.150:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8776 check fall 5 inter 2000 rise 2
```

glance_api

Port number: 9292

Binds to: storage, external

Target network or server: storage on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen glance_api
  bind 172.16.23.250:9292
  bind 172.16.21.250:9292
  server overcloud-controller-0 172.16.21.150:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.21.151:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.21.152:9292 check fall 5 inter 2000 rise 2
```

glance_registry

Port number: 9191

Binds to: internal_api

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen glance_registry
  bind 172.16.20.250:9191
  server overcloud-controller-0 172.16.20.150:9191 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:9191 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:9191 check fall 5 inter 2000 rise 2
```

gnocchi

Port number: 8041

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen gnocchi
  bind 172.16.20.250:8041
  bind 172.16.23.250:8041
  server overcloud-controller-0 172.16.20.150:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8041 check fall 5 inter 2000 rise 2
```

heat_api

Port number: 8004

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.
- This service uses HTTP mode instead of the default TCP mode.

HAProxy example:

```
listen heat_api
  bind 172.16.20.250:8004
  bind 172.16.23.250:8004
  mode http
  server overcloud-controller-0 172.16.20.150:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8004 check fall 5 inter 2000 rise 2
```

heat_cfn

Port number: 8000

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen heat_cfn
  bind 172.16.20.250:8000
  bind 172.16.23.250:8000
  server overcloud-controller-0 172.16.20.150:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.152:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.151:8000 check fall 5 inter 2000 rise 2
```

heat_cloudwatch

Port number: 8003

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen heat_cloudwatch
  bind 172.16.20.250:8003
  bind 172.16.23.250:8003
  server overcloud-controller-0 172.16.20.150:8003 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8003 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8003 check fall 5 inter 2000 rise 2
```

horizon

Port number: 80

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.
- This service uses HTTP mode instead of the default TCP mode.

- This service uses cookie-based persistence for interactions with the UI.

HAProxy example:

```
listen horizon
  bind 172.16.20.250:80
  bind 172.16.23.250:80
  mode http
  cookie SERVERID insert indirect nocache
  server overcloud-controller-0 172.16.20.150:80 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:80 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:80 check fall 5 inter 2000 rise 2
```

keystone_admin

Port number: 35357

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen keystone_admin
  bind 172.16.23.250:35357
  bind 172.16.20.250:35357
  server overcloud-controller-0 172.16.20.150:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:35357 check fall 5 inter 2000 rise 2
```

keystone_admin_ssh

Port number: 22

Binds to: internal_api

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen keystone_admin_ssh
  bind 172.16.20.250:22
  server overcloud-controller-0 172.16.20.150:22 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:22 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:22 check fall 5 inter 2000 rise 2
```

keystone_public

Port number: 5000

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen keystone_public
  bind 172.16.20.250:5000
  bind 172.16.23.250:5000
  server overcloud-controller-0 172.16.20.150:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:5000 check fall 5 inter 2000 rise 2
```

mysql

Port number: 3306

Binds to: internal_api

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check. However, the health checks use port 9200.
- This service is load balanced to only one server at a time.
- Each server is only used in load balancing only when all other non-backup servers are unavailable.
- If the server is offline, all connections are immediately terminated.
- You must enable the sending of TCP keepalive packets on both sides.
- You must enable HTTP protocol to check on the servers health.
- You can configure a stickiness table to store IP addresses, to help maintain persistence.

IMPORTANT

The **mysql** service uses Galera to provide a highly available database cluster. Galera supports an active-active configuration, but to avoid lock contention, you must use an active-passive configuration that is enforced by the load balancer.

HAProxy example:

```
listen mysql
  bind 172.16.20.250:3306
  option tcpka
  option httpchk
  stick on dst
  stick-table type ip size 1000
  timeout client 0
  timeout server 0
  server overcloud-controller-0 172.16.20.150:3306 backup check fall 5 inter 2000 on-marked-down
shutdown-sessions port 9200 rise 2
  server overcloud-controller-1 172.16.20.151:3306 backup check fall 5 inter 2000 on-marked-down
shutdown-sessions port 9200 rise 2
  server overcloud-controller-2 172.16.20.152:3306 backup check fall 5 inter 2000 on-marked-down
shutdown-sessions port 9200 rise 2
```

neutron

Port number: 9696

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen neutron
  bind 172.16.20.250:9696
  bind 172.16.23.250:9696
  server overcloud-controller-0 172.16.20.150:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:9696 check fall 5 inter 2000 rise 2
```

nova_ec2

Port number: 8773

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen nova_ec2
  bind 172.16.20.250:8773
  bind 172.16.23.250:8773
```

```
  server overcloud-controller-0 172.16.20.150:8773 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8773 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8773 check fall 5 inter 2000 rise 2
```

nova_metadata

Port number: 8775

Binds to: internal_api

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen nova_metadata
  bind 172.16.20.250:8775
  server overcloud-controller-0 172.16.20.150:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8775 check fall 5 inter 2000 rise 2
```

nova_novncproxy

Port number: 6080

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.
- The default load balancing method is round-robin. However, for this service, use a **source** method. This method hashes the source IP address and divides it by the total weight of the running servers. This method also designates the server that receives the request and ensures that the same client IP address always reaches the same server unless server goes down or up. If the hash result changes due to a change in the number of running servers, the load balancer redirects the clients to a different server.

HAProxy example:

```
listen nova_novncproxy
  bind 172.16.20.250:6080
  bind 172.16.23.250:6080
  balance source
  server overcloud-controller-0 172.16.20.150:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:6080 check fall 5 inter 2000 rise 2
```

nova_osapi

Port number: 8774

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen nova_osapi
  bind 172.16.20.250:8774
  bind 172.16.23.250:8774
  server overcloud-controller-0 172.16.20.150:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8774 check fall 5 inter 2000 rise 2
```

nova_placement

Port number: 8778

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen nova_placement
  bind 172.16.20.250:8778
  bind 172.16.23.250:8778
  server overcloud-controller-0 172.16.20.150:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8778 check fall 5 inter 2000 rise 2
```

panko

Port number: 8779

Binds to: internal_api, external

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen panko
  bind 172.16.20.250:8779
  bind 172.16.23.250:8779
  server overcloud-controller-0 172.16.20.150:8779 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:8779 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:8779 check fall 5 inter 2000 rise 2
```

redis

Port number: 6379

Binds to: internal_api (redis service IP)

Target network or server: internal_api on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.
- Perform health checks using **tcp-check** send/expect sequences. The string to send is **info\ replication\r\n** and the response is **role:master**.
- The Redis service uses a password for authentication. For example, the HAProxy configuration uses a **tcp-check** with the AUTH method and the Redis administration password. Director normally generates a random password, but you can define a custom Redis password.
- The default balancing method is **round-robin**. However, for this service, use the **first** method. This ensures that the first server that has available connection slots receives the connection.

HAProxy example:

```
listen redis
  bind 172.16.20.249:6379 transparent
  balance first
  option tcp-check
  tcp-check send AUTH\ p@55w0rd!\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-0 172.16.20.150:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.20.151:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.20.152:6379 check fall 5 inter 2000 rise 2
```

swift_proxy_server

Port number: 8080

Binds to: storage, external

Target network or server: storage on overcloud-controller-0, overcloud-controller-1, and overcloud-controller-2

Other information:

- Each target server uses a default health check.

HAProxy example:

```
listen swift_proxy_server
  bind 172.16.23.250:8080
  bind 172.16.21.250:8080
  server overcloud-controller-0 172.16.21.150:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.21.151:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.21.152:8080 check fall 5 inter 2000 rise 2
```

CHAPTER 8. MANAGING DATABASE REPLICATION WITH GALERA

Red Hat OpenStack Platform uses the MariaDB Galera Cluster to manage database replication. Pacemaker runs the Galera service as a bundle set resource that manages the database master/slave status. You can use Galera to test and verify different aspects of the database cluster, such as hostname resolution, cluster integrity, node integrity, and database replication performance.

When you investigate database cluster integrity, each node must meet the following criteria:

- The node is a part of the correct cluster.
- The node can write to the cluster.
- The node can receive queries and write commands from the cluster.
- The node is connected to other nodes in the cluster.
- The node is replicating write-sets to tables in the local database.

8.1. VERIFYING HOSTNAME RESOLUTION IN A MARIADB CLUSTER

To troubleshoot the MariaDB Galera cluster, first eliminate any hostname resolution problems and then check the write-set replication status on the database of each Controller node. To access the MySQL database, use the password set by director during the overcloud deployment.

By default, director binds the Galera resource to a hostname instead of an IP address. Therefore, any problems that prevent hostname resolution, such as misconfigured or failed DNS, might cause Pacemaker to incorrectly manage the Galera resource.

Procedure

1. From a Controller node, get the MariaDB database root password by running the **hiera** command.

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

2. Get the name of the MariaDB container that runs on the node.

```
$ sudo podman ps | grep -i galera
a403d96c5026  undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-mariadb:16.0-
106           /bin/bash /usr/lo...  3 hours ago  Up 3 hours ago         galera-bundle-podman-0
```

3. Get the write-set replication information from the MariaDB database on each node.

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-
PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_%';"
   +----------------------------+----------+
   | Variable_name              | Value    |
   +----------------------------+----------+
   | wsrep_applier_thread_count | 1        |
   | wsrep_apply_oooe           | 0.018672 |
   | wsrep_apply_oool           | 0.000630 |
```

```
| wsrep_apply_window         | 1.021942 |
| ...                        | ...      |
+----------------------------+----------+
```

Each relevant variable uses the prefix **wsrep**.

4. Verify the health and integrity of the MariaDB Galera cluster by checking that the cluster is reporting the correct number of nodes.

8.2. CHECKING MARIADB CLUSTER INTEGRITY

To investigate problems with the MariaDB Galera Cluster, check the integrity of the whole cluster by checking specific **wsrep** database variables on each Controller node.

Procedure

- Run the following command and replace **<variable>** with the **wsrep** database variable that you want to check:

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE <variable>;"
```

The following example shows how to view the cluster state UUID of the node:

```
$ sudo podman exec galera-bundle-podman-0 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 **wsrep** database variables that you can use to check cluster integrity.

Table 8.1. Database variables to check for cluster integrity

Variable	Summary	Description
wsrep_cluster_state_uuid	Cluster state UUID	ID of the cluster to which the node belongs. All nodes must have an identical cluster 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 node. If the value is less than the actual number of nodes, then some nodes either failed or lost connectivity.

Variable	Summary	Description
wsrep_cluster_conf_id	Total number of cluster changes	Determines whether the cluster was split to several components, or partitions. Partitioning is usually caused by a network failure. All nodes must have an identical value. In case some nodes report a different **wsrep_cluster_conf_id**, check the **wsrep_cluster_status** value to see if the nodes can still write to the cluster (**Primary**).
wsrep_cluster_status	Primary component status	Determines whether the node can write to the cluster. If the node can write to the cluster, the **wsrep_cluster_status** value is **Primary**. Any other value indicates that the node is part of a non-operational partition.

8.3. CHECKING DATABASE NODE INTEGRITY IN A MARIADB CLUSTER

To investigate problems with a specific Controller node in the MariaDB Galera Cluster, check the integrity of the node by checking specific **wsrep** database variables.

Procedure

- Run the following command and replace **<variable>** with the **wsrep** database variable that you want to check:

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE <variable>;"
```

The following table lists the **wsrep** database variables that you can use to check node integrity.

Table 8.2. 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** is **ON**.

Variable	Summary	Description
wsrep_connected	Node network connectivity	States whether the node can connect to other nodes on the network. If so, then **wsrep_connected** is **ON**.
wsrep_local_state_comment	Node state	Summarizes the node state. If the node can write to the cluster, then typical values for **wsrep_local_state_comment** can be **Joining**, **Waiting on SST**, **Joined**, **Synced**, or **Donor**. If the node is part of a non-operational component, then the value of **wsrep_local_state_comment** is **Initialized**.

NOTE

- The **wsrep_connected** value can be **ON** even if the node is connected only to a subset of nodes in the cluster. For example, in case of a cluster partition, the node might be part of a component that cannot write to the cluster. For more information about checking cluster integrity, see Section 8.2, "Checking MariaDB cluster integrity".
- If the **wsrep_connected** value is **OFF**, then the node is not connected to any cluster components.

8.4. TESTING DATABASE REPLICATION PERFORMANCE IN A MARIADB CLUSTER

To check the performance of the MariaDB Galera Cluster, run benchmark tests on the replication throughput of the cluster by checking specific **wsrep** database variables.

Every time you query one of these variables, a **FLUSH STATUS** command resets the variable value. To run benchmark tests, you must run multiple queries and analyze the variances. These variances can help you determine how much Flow Control is affecting the cluster performance.

Flow Control is a mechanism that the cluster uses to manage replication. When the local receive queue exceeds a certain threshold, Flow Control pauses the replication until the queue size goes down. For more information about Flow Control, see Flow Control on the Galera Cluster website.

Procedure

- Run the following command and replace **<variable>** with the **wsrep** database variable that you want to check:

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW STATUS LIKE <variable>;"
```

-

The following table lists the **wsrep** database variables that you can use to test database replication performance.

Table 8.3. Database variables to check for database replication performance

Variable	Summary	Usage
wsrep_local_recv_queue_avg	Average size of the local received write-set queue after the last query.	A value higher than **0.0** indicates that the node cannot apply write-sets as quickly as it receives write-sets, which 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	Average send queue length after the last query.	A value higher than **0.0** indicates a higher likelihood of replication throttling and network throughput problems.
wsrep_local_recv_queue_min and **wsrep_local_recv_queue_max**	Minimum and maximum size of the local receive queue after the last query.	If the value of **wsrep_local_recv_queue_avg** is higher than **0.0**, you can check these variables to determine the scope of the queue size.

Variable	Summary	Usage
wsrep_flow_control_paused	Fraction of the time that Flow Control paused the node after the last query.	A value higher than **0.0** indicates that Flow Control paused the node. To determine the duration of the pause, multiply the **wsrep_flow_control_paused** value with the number of seconds between the queries. The optimal value is as close to **0.0** as possible. For example: ● If the value of **wsrep_flow_control_paused** is **0.50** one minute after the last query, then Flow Control paused the node for 30 seconds. ● If the value of **wsrep_flow_control_paused** is **1.0** one minute after the last query, then Flow Control paused the node for the entire minute.
wsrep_cert_deps_distance	Average difference between the lowest and highest sequence number (**seqno**) value that can be applied in parallel	In case of throttling and pausing, this variable indicates how many write-sets on average can be applied in parallel. Compare the value with the **wsrep_slave_threads** variable to see how many write-sets can actually be applied simultaneously.

Variable	Summary	Usage
wsrep_slave_threads	Number of threads that can be applied simultaneously	You can increase the value of this variable to apply more threads simultaneously, which also increases the value of **wsrep_cert_deps_distance**. The value of **wsrep_slave_threads** must not be higher than the number of CPU cores in the node. For example, if the **wsrep_cert_deps_distance** value is **20**, you can increase the value of **wsrep_slave_threads** from **2** to **4** to increase the amount of write-sets that the node can apply. If a problematic node already has an optimal **wsrep_slave_threads** value, you can exclude the node from the cluster while you investigate possible connectivity issues.

8.5. ADDITIONAL RESOURCES

- What is MariaDB Galera Cluster?

CHAPTER 9. TROUBLESHOOTING HIGH AVAILABILITY RESOURCES

In case of resource failure, you must investigate the cause and location of the problem, fix the failed resource, and optionally clean up the resource. There are many possible causes of resource failures depending on your deployment, and you must investigate the resource to determine how to fix the problem.

For example, you can check the resource constraints to ensure that the resources are not interrupting each other, and that the resources can connect to each other. You can also examine a Controller node that is fenced more often than other Controller nodes to identify possible communication problems.

Depending on the location of the resource problem, you choose one of the following options:

Controller node problems

If health checks to a Controller node are failing, this can indicate a communication problem between Controller nodes. To investigate, log in to the Controller node and check if the services can start correctly.

Individual resource problems

If most services on a Controller are running correctly, you can run the **pcs status** command and check the output for information about a specific Pacemaner resource failure or run the **systemctl** command to investigate a non-Pacemaker resource failure.

9.1. VIEWING RESOURCE CONSTRAINTS IN A HIGH AVAILABILITY CLUSTER

Before you investigate resource problems, you can view constraints on how services are launched, including constraints related to where each resource is located, the order in which the resource starts, and whether the resource must be colocated with another resource.

Procedure

- Use one of the following options:
  - To view all resource constraints, log in to any Controller node and run the **pcs constraint show** command:

    ```
    $ sudo pcs constraint show
    ```

    The following example shows a truncated output from the **pcs constraint show** command on a Controller node:

    ```
    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 the following main constraint types:

Location Constraints

Lists the locations to which resources can be assigned:

- The first constraint defines a rule that sets the **galera-bundle** resource to run on nodes with the **galera-role** attribute set to **true**.
- The second location constraint specifies that the IP resource **ip-192.168.24.15** 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 Controller nodes.

Ordering Constraints

Lists the order in which resources can launch. This example shows a constraint that sets the virtual IP address resources **IPaddr2** to start before the HAProxy service.

NOTE

Ordering constraints only apply to IP address resources and to HAproxy. Systemd manages all other resources, because services such as Compute are expected to withstand an interruption of a dependent service, such as Galera.

Colocation Constraints

Lists which resources must be located together. All virtual IP addresses are linked to the **haproxy-bundle** resource.

- To view constraints for a specific resource, log in to any Controller node and run the **pcs property show** command:

```
$ sudo pcs property show
```

Example output:

```
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: tripleo_cluster
 dc-version: 2.0.1-4.el8-0eb7991564
 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
```

In this output, you can verify the that the resource constraints are set correctly. For example, the **galera-role** attribute is **true** for all Controller nodes, which means that the **galera-bundle** resource runs only on these nodes.

9.2. INVESTIGATING PACEMAKER RESOURCE PROBLEMS

To investigate failed resources that Pacemaker manages, log in to the Controller node on which the resource is failing and check the status and log events for the resource. For example, investigate the status and log events for the **openstack-cinder-volume** resource.

Prerequisites

- A Controller node with Pacemaker services
- Root user permissions to view log events

Procedure

1. Log in to the Controller node on which the resource is failing.
2. Run the **pcs status** command with the **grep** option to get the status of the service:

```
# sudo pcs status | grep cinder
  Podman container: openstack-cinder-volume [192.168.24.1:8787/rh-osbs/rhosp161-
openstack-cinder-volume:pcmklatest]
    openstack-cinder-volume-podman-0     (ocf::heartbeat:podman):        Started controller-1
```

3. View the log events for the resource:

```
# sudo less /var/log/containers/stdouts/openstack-cinder-volume.log
[...]
2021-04-12T12:32:17.607179705+00:00 stderr F ++ cat /run_command
```

```
2021-04-12T12:32:17.609648533+00:00 stderr F + CMD='/usr/bin/cinder-volume --config-file
/usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf'
2021-04-12T12:32:17.609648533+00:00 stderr F + ARGS=
2021-04-12T12:32:17.609648533+00:00 stderr F + [[ ! -n '' ]]
2021-04-12T12:32:17.609648533+00:00 stderr F + . kolla_extend_start
2021-04-12T12:32:17.611214130+00:00 stderr F +++ stat -c %U:%G /var/lib/cinder
2021-04-12T12:32:17.616637578+00:00 stderr F ++ [[ cinder:kolla != \c\i\n\d\e\r\:\k\o\l\l\a ]]
2021-04-12T12:32:17.616722778+00:00 stderr F + echo 'Running command:
'\''/usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-dist.conf --config-file
/etc/cinder/cinder.conf'\''
2021-04-12T12:32:17.616751172+00:00 stdout F Running command: '/usr/bin/cinder-volume
--config-file /usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf'
2021-04-12T12:32:17.616775368+00:00 stderr F + exec /usr/bin/cinder-volume --config-file
/usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf
```

4. Correct the failed resource based on the information from the output and from the logs.

5. Run the **pcs resource cleanup** command to reset the status and the fail count of the resource.

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

9.3. INVESTIGATING SYSTEMD RESOURCE PROBLEMS

To investigate failed resources that systemd manages, log in to the Controller node on which the resource is failing and check the status and log events for the resource. For example, investigate the status and log events for the **tripleo_nova_conductor** resource.

Prerequisites

- A Controller node with systemd services
- Root user permissions to view log events

Procedure

1. Run the **systemctl status** command to show the resource status and recent log events:

```
[tripleo-admin@controller-0 ~]$ sudo systemctl status tripleo_nova_conductor
● tripleo_nova_conductor.service - nova_conductor container
   Loaded: loaded (/etc/systemd/system/tripleo_nova_conductor.service; enabled; vendor
preset: disabled)
   Active: active (running) since Mon 2021-04-12 10:54:46 UTC; 1h 38min ago
 Main PID: 5125 (conmon)
    Tasks: 2 (limit: 126564)
   Memory: 1.2M
   CGroup: /system.slice/tripleo_nova_conductor.service
           └─5125 /usr/bin/conmon --api-version 1 -c
cc3c63b54e0864c94ac54a5789be96aea1dd60b2f3216b37c3e020c76e7887d4 -u
cc3c63b54e0864c94ac54a5789be96aea1dd60b2f3216b37c3e020c76e7887d4 -r
/usr/bin/runc -b /var/lib/containers/storage/overlay-
containers/cc3c63b54e0864c94ac54a5789be96aea1dd60b2f3216b37c3e02>
```

```
Apr 12 10:54:42 controller-0.redhat.local systemd[1]: Starting nova_conductor container...
Apr 12 10:54:46 controller-0.redhat.local podman[2855]: nova_conductor
Apr 12 10:54:46 controller-0.redhat.local systemd[1]: Started nova_conductor container.
```

2. View the log events for the resource:

```
# sudo less /var/log/containers/tripleo_nova_conductor.log
```

3. Correct the failed resource based on the information from the output and from the logs.

4. Restart the resource and check the status of the service:

```
# systemctl restart tripleo_nova_conductor
# systemctl status tripleo_nova_conductor
● tripleo_nova_conductor.service - nova_conductor container
   Loaded: loaded (/etc/systemd/system/tripleo_nova_conductor.service; enabled; vendor
preset: disabled)
   Active: active (running) since Thu 2021-04-22 14:28:35 UTC; 7s ago
  Process: 518937 ExecStopPost=/usr/bin/podman stop -t 10 nova_conductor (code=exited,
status=0/SUCCESS)
  Process: 518653 ExecStop=/usr/bin/podman stop -t 10 nova_conductor (code=exited,
status=0/SUCCESS)
  Process: 519063 ExecStart=/usr/bin/podman start nova_conductor (code=exited,
status=0/SUCCESS)
 Main PID: 519198 (conmon)
    Tasks: 2 (limit: 126564)
   Memory: 1.1M
   CGroup: /system.slice/tripleo_nova_conductor.service
           └─519198 /usr/bin/conmon --api-version 1 -c
0d6583beb20508e6bacccd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -u
0d6583beb20508e6bacccd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -r /usr/bin/runc
-b /var/lib/containe>

Apr 22 14:28:34 controller-0.redhat.local systemd[1]: Starting nova_conductor container...
Apr 22 14:28:35 controller-0.redhat.local podman[519063]: nova_conductor
Apr 22 14:28:35 controller-0.redhat.local systemd[1]: Started nova_conductor container.
```

CHAPTER 10. MONITORING A HIGH AVAILABILITY RED HAT CEPH STORAGE CLUSTER

When you deploy an overcloud with Red Hat Ceph Storage, Red Hat OpenStack Platform uses the **ceph-mon** monitor daemon to manage the Ceph cluster. Director deploys the daemon on all Controller nodes.

10.1. CHECKING RED HAT CEPH MONITORING SERVICE STATUS

To check the status of the Red Hat Ceph Storage monitoring service, log in to a Controller node and run the **service ceph status** command.

Procedure

- Log in to a Controller node and check that the Ceph Monitoring service is running:

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

10.2. CHECKING RED HAT CEPH MONITORING CONFIGURATION

To check the configuration of the Red Hat Ceph Storage monitoring service, log in to a Controller node or a Red Hat Ceph node and open the **/etc/ceph/ceph.conf** file.

Procedure

- Log in to a Controller nodes or on a Ceph node and open the **/etc/ceph/ceph.conf** file to view the monitoring configuration parameters:

```
[global]
osd_pool_default_pgp_num = 128
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
```

This example shows the following information:

  - All three Controller nodes are configured to monitor the Red Hat Ceph Storage cluster with the **mon_initial_members** parameter.
  - The **172.19.0.11/24** network is configured to provide a communication path between the Controller nodes and the Red Hat Ceph Storage nodes.

- The Red Hat Ceph Storage nodes are assigned to a separate network from the Controller nodes, and the IP addresses for the monitoring Controller nodes are **172.18.0.15**, **172.18.0.16**, and **172.18.0.17**.

10.3. CHECKING RED HAT CEPH NODE STATUS

To check the status of a specific Red Hat Ceph Storage node, log in to the node and run the **ceph -s** command.

Procedure

- Log in to the Ceph node and run the **ceph -s** 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
```

This example output shows that the **health** parameter value is **HEALTH_OK**, which indicates that the Ceph node is active and healthy. The output also shows three Ceph monitor services that are running on the three **overcloud-controller** nodes and the IP addresses and ports of the services.

10.4. ADDITIONAL RESOURCES

- Red Hat Ceph product page