Red Hat OpenStack Platform 16.1

High Availability Deployment and Usage

Planning, deploying, and managing high availability in Red Hat OpenStack Platform
Planning, deploying, and managing high availability in Red Hat OpenStack Platform

OpenStack Team
rhos-docs@redhat.com
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 .................................................. 3

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .......................... 4

CHAPTER 1. HIGH AVAILABILITY SERVICES ........................................... 5

CHAPTER 2. EXAMPLE DEPLOYMENT: HIGH AVAILABILITY CLUSTER WITH COMPUTE AND CEPH ....... 6
  2.1. HARDWARE SPECIFICATIONS ................................................. 7
  2.2. NETWORK SPECIFICATIONS ............................................... 8
  2.3. UNDERCLOUD CONFIGURATION FILES .................................. 9
  2.4. OVERCLOUD CONFIGURATION FILES .................................... 12

CHAPTER 3. ACCESSING THE HIGH AVAILABILITY ENVIRONMENT ................ 18

CHAPTER 4. MANAGING HIGH AVAILABILITY SERVICES WITH PACEMAKER ........ 19
  4.1. RESOURCE BUNDLES AND CONTAINERS .................................. 19
  4.2. VIEWING GENERAL PACEMAKER INFORMATION ....................... 22
  4.3. VIEWING BUNDLE STATUS ............................................... 23
  4.4. VIEWING VIRTUAL IP ADDRESSES ...................................... 23
  4.5. VIEWING PACEMAKER STATUS AND POWER MANAGEMENT INFORMATION 26
  4.6. TROUBLESHOOTING FAILED PACEMAKER RESOURCES ................. 27

CHAPTER 5. FENCING CONTROLLER NODES WITH STONITH ........................ 28
  5.1. SUPPORTED FENCING AGENTS ........................................... 28
  5.2. DEPLOYING FENCING ON THE OVERCLOUD ............................ 29
  5.3. TESTING FENCING ON THE OVERCLOUD ................................ 32
  5.4. VIEWING STONITH INFORMATION ...................................... 33
  5.5. FENCING PARAMETERS ................................................... 34

CHAPTER 6. LOAD BALANCING TRAFFIC WITH HAPROXY ........................... 36
  6.1. HOW HAPROXY WORKS .................................................. 36
  6.2. VIEWING HAPROXY STATS ............................................... 37

CHAPTER 7. MANAGING DATABASE REPLICATION WITH GALERA ..................... 38
  7.1. VERIFYING HOSTNAME RESOLUTION ................................... 38
  7.2. CHECKING DATABASE CLUSTER INTEGRITY ............................ 39
  7.3. CHECKING DATABASE NODE INTEGRITY ................................ 40
  7.4. TESTING DATABASE REPLICATION PERFORMANCE ..................... 41

CHAPTER 8. TROUBLESHOOTING RESOURCE PROBLEMS ............................. 44
  8.1. VIEWING RESOURCE CONSTRAINTS ..................................... 44
  8.2. INVESTIGATING PACEMAKER RESOURCE PROBLEMS .................. 46
  8.3. INVESTIGATING SYSTEMD RESOURCE PROBLEMS ...................... 47

CHAPTER 9. MONITORING A HIGH AVAILABILITY RED HAT CEPH STORAGE CLUSTER .... 49
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.

Using the Direct Documentation Feedback (DDF) function

Use the Add Feedback DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the Multi-page HTML format.
2. Ensure that you see the Feedback button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click Add Feedback.
5. Complete the Add Feedback field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click Submit.
CHAPTER 1. HIGH AVAILABILITY SERVICES

Red Hat OpenStack Platform (RHOSP) employs several technologies to provide the services required to implement high availability (HA).

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 with the director, the 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.

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

The following example scenario shows the architecture, hardware and network specifications, as well as 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

For more information about deploying a Red Hat Ceph Storage cluster, see Deploying an Overcloud with Containerized Red Hat Ceph.

For more information about deploying Red Hat OpenStack Platform with director, see Director Installation and Usage.
2.1. HARDWARE SPECIFICATIONS

The following table shows the hardware used in the example deployment. You can adjust the CPU, memory, storage, or NICs as needed in your own test deployment.

Table 2.1. Physical computers

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

Review the following guidelines before you plan hardware assignments:

**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.

2.2. NETWORK SPECIFICATIONS

The following table shows the network configuration used in the example deployment.

Table 2.2. Physical and virtual networks

<table>
<thead>
<tr>
<th>Physical NICs</th>
<th>Purpose</th>
<th>VLANs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth0</td>
<td>Provisioning network (undercloud)</td>
<td>N/A</td>
<td>Manages all nodes from director (undercloud)</td>
</tr>
<tr>
<td>eth1 and eth2</td>
<td>Controller/External (overcloud)</td>
<td>N/A</td>
<td>Bonded NICs with VLANs</td>
</tr>
<tr>
<td></td>
<td>External network</td>
<td>VLAN 100</td>
<td>Allows access from outside the environment to the project networks, internal API, and OpenStack Horizon Dashboard</td>
</tr>
<tr>
<td></td>
<td>Internal API</td>
<td>VLAN 201</td>
<td>Provides access to the internal API between Compute nodes and Controller nodes</td>
</tr>
<tr>
<td></td>
<td>Storage access</td>
<td>VLAN 202</td>
<td>Connects Compute nodes to storage media</td>
</tr>
<tr>
<td></td>
<td>Storage management</td>
<td>VLAN 203</td>
<td>Manages storage media</td>
</tr>
<tr>
<td></td>
<td>Project network</td>
<td>VLAN 204</td>
<td>Provides project network services to RHOSP</td>
</tr>
</tbody>
</table>

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.

2.3. UNDERCLOUD CONFIGURATION FILES

The example deployment uses the following undercloud configuration files.

instackenv.json

```json
{
    "nodes": [
    {
        "pm_password": "testpass",
        "memory": "24",
        "pm_addr": "10.100.0.11",
        "mac": [
            "2c:c2:60:3b:b3:94"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
    },
    {
        "pm_password": "testpass",
        "memory": "24",
        "pm_addr": "10.100.0.12",
        "mac": [
            "2c:c2:60:51:b7:fb"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
    },
    {
        "pm_password": "testpass",
        "memory": "24",
        "pm_addr": "10.100.0.13",
        "mac": [
            "2c:c2:60:76:ce:a5"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
    },
    {
        "pm_password": "testpass",
        "memory": "24",
        "pm_addr": "10.100.0.14",
        "mac": [
            "2c:c2:60:76:ce:a5"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
    }
}
"pm_password": "testpass",
"memory": "24",
"pm_addr": "10.100.0.51",
"mac": [
  "2c:c2:60:08:b1:e2"
],
"pm_type": "pxe_ipmitool",
"disk": "40",
"arch": "x86_64",
"cpu": "1",
"pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.52",
  "mac": [
    "2c:c2:60:20:a1:9e"
  ],
  "pm_type": "pxe_ipmitool",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "1",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.53",
  "mac": [
    "2c:c2:60:58:10:33"
  ],
  "pm_type": "pxe_ipmitool",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "1",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.101",
  "mac": [
    "2c:c2:60:31:a9:55"
  ],
  "pm_type": "pxe_ipmitool",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "2",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.102",
  "mac": [
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: "br-ex"
# Customize bonding options if required
BondInterfaceOvsOptions:
"bond_mode=active-backup lacp=off other_config:bond-miimon-interval=100"

2.4. OVERCLOUD CONFIGURATION FILES

The example deployment uses the following overcloud configuration files.

/var/lib/config-data/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
option 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

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
option 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\n
tcp-check send PING\n
tcp-check expect string +PONG

tcp-check send info\ replication\n
tcp-check expect string role:master

tcp-check send QUIT\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
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```
CHAPTER 3. ACCESSING THE HIGH AVAILABILITY ENVIRONMENT

You can access and investigate specific HA nodes from the undercloud.

Procedure

1. In a running HA environment, log in to the undercloud.

2. Change to the `stack` user:
   
   ```
   # sudo su - stack
   ```

3. To access and investigate an undercloud node, get the IP address of the node from the undercloud:
   
   ```
   (undercloud) $ source ~/stackrc
   (undercloud) $ openstack server list
   +-------+------------------------+---+----------------------+---+
   | ID    | Name                   |...| Networks             |...|
   +-------+------------------------+---+----------------------+---+
   | d1... | overcloud-controller-0 |...| ctlplane=*10.200.0.11* |...|
   ```

4. To log in to one of the overcloud nodes, run the following commands:
   
   ```
   (undercloud) $ source ~/overcloudrc
   (overcloud) $ ssh [NODE_NAME]@[NODE_IP]
   ```

Replace the name and IP address with the actual values from your deployment.
CHAPTER 4. 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.

For more information about Pacemaker in Red Hat Enterprise Linux, see Configuring and Managing High Availability Clusters in the Red Hat Enterprise Linux documentation.

4.1. 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.

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:


- haproxy-bundle-podman-0 (ocf::heartbeat:podman): Started overcloud-controller-0

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:

```bash
$ sudo pcs resource show haproxy-bundle
Bundle: haproxy-bundle
    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=/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. To check the `haproxy` configuration from the host, open 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.

**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:

```
rabbitmq-bundle-0 (ocf::heartbeat:rabbitmq-cluster): Started overcloud-controller-0

galera-bundle-0 (ocf::heartbeat:galera): Master overcloud-controller-0
galera-bundle-1 (ocf::heartbeat:galera): Master overcloud-controller-1

redis-bundle-0 (ocf::heartbeat:redis): Master overcloud-controller-0
redis-bundle-1 (ocf::heartbeat:redis): Slave overcloud-controller-1
```

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
  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=/etc/hosts target-dir=/etc/hosts (mysql-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (mysql-localtime)
  options=ro source-dir=/var/lib/mysql target-dir=/var/lib/mysql (mysql-lib)
  options=ro source-dir=/var/log/mariadb target-dir=/var/log/mariadb (mysql-log-mariadb)
  options=ro 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
```
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 4.4, “Viewing virtual IP addresses”.

4.2. VIEWING GENERAL PACEMAKER INFORMATION

To view general Pacemaker information, use the pcs status command.

Procedure

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

   $ ssh heat-admin@overcloud-controller-0

2. Run the pcs status command:

   [heat-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 4.1, “Resource bundles and containers”.

### 4.3. VIEWING BUNDLE STATUS

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.

#### Check bundle status from an undercloud node

Run the following command:

```bash
$ 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.

#### Check bundle status from a Controller node

Log in to a Controller node and run the following command:

```bash
$ 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
```

### 4.4. VIEWING VIRTUAL IP ADDRESSES

Each IPaddr2 resource sets a virtual IP address that clients use to request access to a service.
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.

**Show all virtual IP addresses**

Run the `pcs resource show` command with the `--full` option to display all resources that use the `VirtualIP` type:

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

The following example output shows each Controller node that is currently set to listen to a particular virtual IP address:

```
ip-10.200.0.6 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
ip-192.168.1.150 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
ip-172.16.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
ip-172.16.0.11 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
ip-172.18.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-2
```

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 4.1. IP address description and allocation source**

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Description</th>
<th>Allocated From</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.150</td>
<td>Public IP address</td>
<td><strong>ExternalAllocationPools</strong> attribute in the <code>network-environment.yaml</code> file</td>
</tr>
<tr>
<td>10.200.0.6</td>
<td>Controller virtual IP address</td>
<td>Part of the <code>dhcp_start</code> and <code>dhcp_end</code> range set to <code>10.200.0.5-10.200.0.24</code> in the <code>undercloud.conf</code> file</td>
</tr>
<tr>
<td>172.16.0.10</td>
<td>Provides access to OpenStack API services on a Controller node</td>
<td><strong>InternalApiAllocationPools</strong> in the <code>network-environment.yaml</code> file</td>
</tr>
<tr>
<td>172.18.0.10</td>
<td>Storage virtual IP address that provides access to the Glance API and to Swift Proxy services</td>
<td><strong>StorageAllocationPools</strong> attribute in the <code>network-environment.yaml</code> file</td>
</tr>
<tr>
<td>172.16.0.11</td>
<td>Provides access to Redis service on a Controller node</td>
<td><strong>InternalApiAllocationPools</strong> in the <code>network-environment.yaml</code> file</td>
</tr>
</tbody>
</table>
### View a specific IP address

Run the **pcs resource show** command.

```bash
$ 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)
```

### View network information for a specific IP address

1. Log in to the Controller node that is assigned to the IP address you want to view.

2. Run the **ip addr show** command to view network interface information.

```bash
$ 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
```

3. Run the **netstat** command to show all processes that listen to the IP address.

```bash
$ 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 0 192.168.1.150:8037    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
```
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.

View port number assignments

Open the /var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg file to see default port number assignments.

The following example shows the port numbers and the services that they listen to:

- 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.

4.5. VIEWING PACEMAKER STATUS AND POWER MANAGEMENT INFORMATION

The last sections of the pcs status output show information about your power management fencing, such as IPMI, and the status of the Pacemaker service itself:

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

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

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

4.6. TROUBLESHOOTING FAILED PACEMAKER RESOURCES

If one the Pacemaker resources fails, you can view the Failed Actions section of the pcs status output.

In the following example, the openstack-cinder-volume service stopped working on controller-0:

```
Failed Actions:
* openstack-cinder-volume_monitor_60000 on overcloud-controller-0 'not running' (7): call=74, status=complete, exitreason='none', last-rc-change='Wed Dec 14 08:33:14 2016', queued=0ms, exec=0ms
```

In this case, you must enable the systemd service openstack-cinder-volume. In other cases, you might need to locate and fix the problem and then clean up the resources. For more information about troubleshooting resource problems, see Chapter 8, Troubleshooting resource problems.
CHAPTER 5. 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".

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.

STONITH is disabled by default and requires manual configuration so that Pacemaker can control the power management of each node in the cluster.

**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.

For more information on fencing with Pacemaker in Red Hat Enterprise Linux, see "Configuring fencing in a Red Hat High Availability cluster" in the Red Hat Enterprise Linux 8 Configuring and Managing High Availability Clusters guide.

5.1. SUPPORTED FENCING AGENTS

When you deploy a high availability environment with fencing, you can choose one of the following fencing agents based on your environment needs. To change the fencing agent, you must configure additional parameters in the fencing.yaml file. For more information, see Section 5.2, "Deploying fencing on the overcloud".

**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](#).

**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 [DMTF Documentation](#).

**fence_rhevm** for Red Hat Virtualization (RHV)

Use to configure fencing for Controller nodes that run in RHV environments. You can generate the `fencing.yaml` file in the same way as you do for IPMI fencing, but you must define the `pm_type` parameter in the `nodes.json` file to use RHV.

By default, the `ssl_insecure` parameter is set to accept self-signed certificates. You can change the parameter value based on your security requirements.

**IMPORTANT**

Ensure that you use a role that has permissions to create and launch virtual machines in RHV, such as `UserVMManager`.

**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.

### 5.2. 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 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 5.1, “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.

- If you use the Red Hat Virtualization (RHV) fencing agent, use a role that has permissions to manage virtual machines, such as `UserVMManager`.

Procedure

1. Log in to each Controller node as the `heat-admin` user.

2. Verify that the cluster is running:

   ```bash
   $ sudo pcs status
   ```

   Example output:

   ```
   Cluster name: openstackHA
   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:

   ```bash
   $ 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:

     ```bash
     $ 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 stack user and check that Pacemaker is configured as the resource manager:

    $ source stackrc
$ openstack server list | grep controller
$ ssh heat-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 5.3, “Testing fencing on the overcloud”
- Section 5.5, “Fencing parameters”
- Exploring RHEL High Availability’s Components - sbd and fence_sbd

5.3. 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 5.2, “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 heat-admin@<controller-x_ip>

2. Change to the root user and close all connections to the Controller node:
3. From a different Controller node, locate the fencing event in the Pacemaker log file:

```bash
$ ssh heat-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.

5.4. VIEWING STONITH INFORMATION

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

```bash
$ sudo pcs stonith show --full
```

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

The `--full` option returns fencing details about the three Controller nodes.

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.

5.5. FENCING PARAMETERS

When you deploy fencing on the overcloud, you generate a fencing.yaml file with the required parameters to configure fencing. For more information about deploying and testing fencing, see Section 5.2, “Deploying fencing on the overcloud”.

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. For a list of supported fencing agents, see Section 5.1, “Supported fencing agents”.
- **host_mac**: Unique identifier for the fencing device.
- **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.
CHAPTER 6. 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.

For more information about HAProxy, see the HAProxy 1.8 documentation.

For information on verifying that HAProxy is configured correctly, see the KCS article How can I verify my haproxy.cfg is correctly configured to load balance openstack services?

6.1. 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/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>
<thead>
<tr>
<th>aodh</th>
<th>cinder</th>
<th>glance_api</th>
<th>gnocchi</th>
</tr>
</thead>
<tbody>
<tr>
<td>haproxy.stats</td>
<td>heat_api</td>
<td>heat_cfn</td>
<td>horizon</td>
</tr>
<tr>
<td>keystone_admin</td>
<td>keystone_public</td>
<td>mysql</td>
<td>neutron</td>
</tr>
<tr>
<td>nova_metadata</td>
<td>nova_novncproxy</td>
<td>nova_osapi</td>
<td>nova_placement</td>
</tr>
</tbody>
</table>

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
```
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.

### 6.2. 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.

**Procedure**

1. Open the `/var/lib/config-data/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.
CHAPTER 7. 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.

Similar to other Pacemaker services, you can use the pcs status command to check that the Galera service is running, and on which Controller nodes it is running. For more information about viewing Pacemaker bundle status, see Section 4.3, “Viewing bundle status”.

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.

7.1. VERIFYING HOSTNAME RESOLUTION

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.

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

**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    |
   +-----------------+----------+
   ```
4. Verify the health and integrity of the MariaDB Galera cluster by first checking that the cluster is reporting the correct number of nodes.

### 7.2. CHECKING DATABASE CLUSTER INTEGRITY

When you investigate problems with the MariaDB Galera Cluster, you can 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';"
```

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsrep_cluster_state_uuid</td>
<td>e2c9a15e-5485-11e0-0800-6bbb637e7211</td>
</tr>
</tbody>
</table>
```

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsrep_cluster_state_uuid</td>
<td>Cluster state UUID</td>
<td>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.</td>
</tr>
<tr>
<td>wsrep_cluster_size</td>
<td>Number of nodes in the cluster</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Variable Summary Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wsrep_cluster_conf_id</code></td>
<td>Total number of cluster changes</td>
<td>Determines whether the cluster was split to several components, also known as <em>partitions</em>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Partitioning is usually caused by a network failure. All nodes must have an identical value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In case some nodes report a different <code>wsrep_cluster_conf_id</code>, check the <code>wsrep_cluster_status</code> value to see if the nodes can still write to the cluster <em>(Primary)</em>.</td>
</tr>
<tr>
<td><code>wsrep_cluster_status</code></td>
<td>Primary component status</td>
<td>Determines whether the node can write to the cluster. If the node can write to the cluster, the <code>wsrep_cluster_status</code> value is <em>Primary</em>. Any other value indicates that the node is part of a non-operational partition.</td>
</tr>
</tbody>
</table>

### 7.3. CHECKING DATABASE NODE INTEGRITY

If you can isolate a Galera cluster problem to a specific node, certain `wsrep` database variables can indicate the specific problem in the 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 table lists the `wsrep` database variables that you can use to check node integrity.

**Table 7.2. Database variables to check for node integrity**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wsrep_ready</code></td>
<td>Node ability to accept queries</td>
<td>States whether the node can accept write-sets from the cluster. If so, then <code>wsrep_ready</code> is ON.</td>
</tr>
<tr>
<td><code>wsrep_connected</code></td>
<td>Node network connectivity</td>
<td>States whether the node can connect to other nodes on the network. If so, then <code>wsrep_connected</code> is ON.</td>
</tr>
</tbody>
</table>
**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 7.2, "Checking database cluster integrity".
- If the **wsrep_connected** value is **OFF**, then the node is not connected to any cluster components.

### 7.4. TESTING DATABASE REPLICATION PERFORMANCE

If the cluster and the individual nodes are all healthy and stable, you can run performance benchmark tests on the replication throughput by querying specific 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’s performance.

Flow Control is a mechanism that the cluster uses to manage replication. When the local received 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:

```bash
$ 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>
<thead>
<tr>
<th>Variable</th>
<th>Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wsrep_local_state_comment</strong></td>
<td>Node state</td>
<td>Summarizes the node state. If the node can write to the cluster, then typical values for <strong>wsrep_local_state_comment</strong> can be <strong>Joining</strong>, <strong>Waiting on SST</strong>, <strong>Joined</strong>, <strong>Synced</strong>, or <strong>Donor</strong>. If the node is part of a non-operational component, then the value of <strong>wsrep_local_state_comment</strong> is <strong>Initialized</strong>.</td>
</tr>
<tr>
<td>Variable</td>
<td>Summary</td>
<td>Usage</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td><code>wsrep_local_recv_queue_avg</code></td>
<td>Average size of the local received write-set queue after the last query.</td>
<td>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 <code>wsrep_local_recv_queue_min</code> and <code>wsrep_local_recv_queue_max</code> for a detailed look at this benchmark.</td>
</tr>
<tr>
<td><code>wsrep_local_send_queue_avg</code></td>
<td>Average send queue length after the last query.</td>
<td>A value higher than 0.0 indicates a higher likelihood of replication throttling and network throughput problems.</td>
</tr>
<tr>
<td><code>wsrep_local_recv_queue_min</code> and <code>wsrep_local_recv_queue_max</code></td>
<td>Minimum and maximum size of the local receive queue after the last query.</td>
<td>If the value of <code>wsrep_local_recv_queue_avg</code> is higher than 0.0, you can check these variables to determine the scope of the queue size.</td>
</tr>
</tbody>
</table>
| `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. |
<table>
<thead>
<tr>
<th>Variable</th>
<th>Summary</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wsrep_cert_deps_distance</strong></td>
<td>Average difference between the lowest and highest sequence number (seqno) value that can be applied in parallel</td>
<td>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 <strong>wsrep_slave_threads</strong> variable to see how many write-sets can actually be applied simultaneously.</td>
</tr>
<tr>
<td><strong>wsrep_slave_threads</strong></td>
<td>Number of threads that can be applied simultaneously</td>
<td>You can increase the value of this variable to apply more threads simultaneously, which also increases the value of <strong>wsrep_cert_deps_distance</strong>. The value of <strong>wsrep_slave_threads</strong> must not be higher than the number of CPU cores in the node. For example, if the <strong>wsrep_cert_deps_distance</strong> value is 20, you can increase the value of <strong>wsrep_slave_threads</strong> from 2 to 4 to increase the amount of write-sets that the node can apply. If a problematic node already has an optimal <strong>wsrep_slave_threads</strong> value, you can exclude the node from the cluster while you investigate possible connectivity issues.</td>
</tr>
</tbody>
</table>
CHAPTER 8. TROUBLESHOOTING RESOURCE PROBLEMS

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.

Typically you perform the following actions to investigate and fix resource problems:

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.

8.1. VIEWING RESOURCE CONSTRAINTS

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.

View all resource constraints

On any Controller node, 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
      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.

View Galera location constraints
On any Controller node, run the pcs property show command.

$ sudo pcs property show

Example output:

Cluster Properties:
cluster-infrastructure: corosync
In this output, you can verify that the \texttt{galera-role} attribute is \texttt{true} for all Controller nodes. This means that the \texttt{galera-bundle} resource runs only on these nodes. The same concept applies to the other attributes associated with the other location constraints.

8.2. 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 \texttt{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 \texttt{pcs status} command with the \texttt{grep} option to get the status of the service:

   ```
   # sudo pcs status | grep cinder
   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="
   2021-04-12T12:32:17.611214130+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
   2021-04-12T12:32:17.616722778+00:00 stderr F + echo 'Running command: \\
   2021-04-12T12:32:17.61722778+00:00 stderr F + echo '/
   ```
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
```

## 8.3. 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:

   ```
   [heat-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 cc3c63b54e0864c94ac54a5789be96aae1dd60b2f3216b37c3e020c76e7887d4 -u cc3c63b54e0864c94ac54a5789be96aae1dd60b2f3216b37c3e020c76e7887d4 -r /usr/bin/runc -b /var/lib/containers/storage/overlay-containers/cc3c63b54e0864c94ac54a5789be96aae1dd60b2f3216b37c3e02>
   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 
            0d6583beb20508e6baccd5fe949471207cb7d4650fec5f3638c2ce6 -u
            0d6583beb20508e6baccd5fe949471207cb7d4650fec5f3638c2ce6 -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 9. 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.

View the status of the Ceph Monitoring service

On a Controller node, run the *service ceph status* command to 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"}
```

View Ceph Monitoring configuration

On a Controller nodes or on a Ceph node, open the `/etc/ceph/ceph.conf` file to view the monitoring configuration parameters:

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

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`.

View individual Ceph node status

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

```
# ceph -s
  cluster 8c835acc-6838-11e5-bb96-2cc260178a92
  health HEALTH_OK
```
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.

For more information about Red Hat Ceph Storage, see the [Red Hat Ceph product page](https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/5/html).