Red Hat OpenStack Platform 16.2

高可用性部署和使用量

在 Red Hat OpenStack Platform 中规划、部署和管理高可用性

Last Updated: 2024-02-07

Red Hat OpenStack Platform 16.2 高可用性部署和使用量

在 Red Hat OpenStack Platform 中规划、部署和管理高可用性

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

为了保持 OpenStack 环境高效启动并运行，请使用 Red Hat OpenStack Platform director 创建跨 OpenStack 中所有主要服务的高可用性和负载均衡的配置。

目录

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

在 JIRA 中提供文档反馈

使用 Create Issue 表单对文档提供反馈。JIRA 问题将在 Red Hat OpenStack Platform Jira 项目中创建，您可以在其中跟踪您的反馈进度。

1. 确保您已登录到 JIRA。如果您没有 JIRA 帐户，请创建一个帐户来提交反馈。
2. 点击以下链接打开 **Create Issue** 页面： Create Issue
3. 完成 **Summary** 和 **Description** 字段。在 **Description** 字段中，包含文档 URL、章节或章节号以及问题的详细描述。不要修改表单中的任何其他字段。
4. 点 **Create**。

第 1 章 RED HAT OPENSTACK PLATFORM 高可用性概述和规划

Red Hat OpenStack Platform (RHOSP)高可用性(HA)是为部署编配故障转移和恢复的服务集合。在规划 HA 部署时，请确保检查环境的不同方面的注意事项，如硬件分配和网络配置。

1.1. RED HAT OPENSTACK PLATFORM 高可用性服务

Red Hat OpenStack Platform (RHOSP)使用一些技术来提供实现高可用性(HA)所需的服务。这些服务包括 Galera、Raba、Redis、HAProxy、Pacemaker 管理的独立服务，以及 Podman 管理的普通容器服务。

1.1.1. 服务类型

Core 容器

核心容器服务包括 Galera、rabbitmq、Redis 和 HAProxy。这些服务在所有 Controller 节点上运行，需要启动、停止和重启操作的特定管理和约束。您可以使用 Pacemaker 启动、管理并对核心容器服务进行故障排除。

注意

RHOSP 使用 MariaDB Galera 集群 来管理数据库复制。

Active-passive

主动 - 被动服务在一个 Controller 节点上运行，并包括 **openstack-cinder-volume** 等服务。要移动主动 - 被动服务，您必须使用 Pacemaker 来确保遵循正确的 stop-start 序列。

systemd 和普通容器

systemd 和普通容器服务是可中断服务的独立服务。因此，如果您重新启动了一个高可用性服务（如 Galera），则不需要手动重启任何其他服务，如 **nova-api**。您可以使用 systemd 或 Podman 直接管理 systemd 和普通容器服务。
在编排 HA 部署时，director 使用 templates 和 Puppet 模块来确保正确配置并启动所有服务。另外，当对 HA 问题进行故障排除时，您必须使用 **podman** 命令或 **systemctl** 命令与 HA 框架中的服务交互。

1.1.2. 服务模式

HA 服务可以以以下模式之一运行：

Active-active

Pacemaker 在多个 Controller 节点上运行相同的服务，并使用 HAProxy 在节点间或通过单个 IP 地址向特定控制器分发流量。在某些情况下，HAProxy 将流量分发到具有 Round945in 调度的活动服务。您可以添加更多 Controller 节点以提高性能。

重要

主动-主动模式只在边缘站点的分布式计算节点(DCN)架构中支持。

Active-passive

无法在主动-主动模式下运行的服务必须以主动-被动模式运行。在这个模式中，服务只有一个实例一次处于活跃状态。例如，HAProxy 使用粘滞性选项将传入的 Galera 数据库连接请求定向到单一后端服务。这有助于防止多个与来自多个 Galera 节点的数据同时连接。

1.2. 规划高可用性硬件分配

在规划硬件分配时，请考虑要在部署中运行的节点数，以及您计划在 Compute 节点上运行的虚拟机(vm)实例数量。

Controller 节点

大多数非存储服务都在 Controller 节点上运行。所有服务在三个节点之间复制，并配置为主动或主动 - 被动服务。高可用性(HA)环境至少需要三个节点。

Red Hat Ceph Storage 节点

存储服务在这些节点上运行，并为 Compute 节点提供 Red Hat Ceph Storage 区域池。至少需要三个节点。

Compute 节点

虚拟机(VM)实例在 Compute 节点上运行。您可以根据需要部署任意数量的 Compute 节点来满足容量要求，以及迁移和重启操作。您必须将 Compute 节点连接到存储网络和项目网络，以确保虚拟机能够访问存储节点、其他 Compute 节点上的虚拟机和公共网络。

STONITH

您必须在高可用性 overcloud 中为作为 Pacemaker 集群一部分的每个节点配置 STONITH 设备。不支持在不使用 STONITH 的情况下部署高可用性 overcloud。有关 STONITH 和 Pacemaker 的信息，请参阅红帽高可用性集群中的隔离和 RHEL 高可用性集群的支持策略。

1.3. 规划高可用性网络

当您计划虚拟网络和物理网络时，请考虑 置备网络交换机配置和外部网络交换机配置。

除了网络配置外，还必须部署以下组件：

置备网络交换机

- 此交换机必须能够将 undercloud 连接到 overcloud 中的所有物理机。
- 连接到此交换机的每个 overcloud 节点上的 NIC 必须能够从 undercloud 进行 PXE 引导。
- 必须启用 **portfast** 参数。

Controller/External 网络交换机

- 此交换机必须配置为为部署中的其他 VLAN 执行 VLAN 标记。
- 仅允许 VLAN 100 流量到外部网络。

网络硬件和 keystone 端点

- 为防止 Controller 节点网卡或网络交换机故障破坏 overcloud 服务可用性，请确保 keystone 管理端点位于使用绑定网卡或网络硬件冗余的网络中。
  如果将 Keystone 端点移到不同的网络，如 **internal_api**，请确保 undercloud 可以访问 VLAN 或子网。有关更多信息，请参阅红帽知识库解决方案如何将 Keystone 管理端点迁移到 internal_api 网络。

1.4. 访问高可用性环境

要调查高可用性(HA)节点，请使用 **stack** 用户登录 overcloud 节点，并运行 **openstack server list** 命令来查看节点的状态和详情。

先决条件

- 高可用性已部署并运行。

流程

1. 在正在运行的 HA 环境中，以 **stack** 用户身份登录 undercloud。
2. 确定 overcloud 节点的 IP 地址：

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

3. 登录到其中一个 overcloud 节点：

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

将 **<node_ip** > 替换为您要登录到的节点的 IP 地址。

1.5. 其他资源

- *第 2 章 示例部署：使用 Compute 和 Ceph 的高可用性集群*

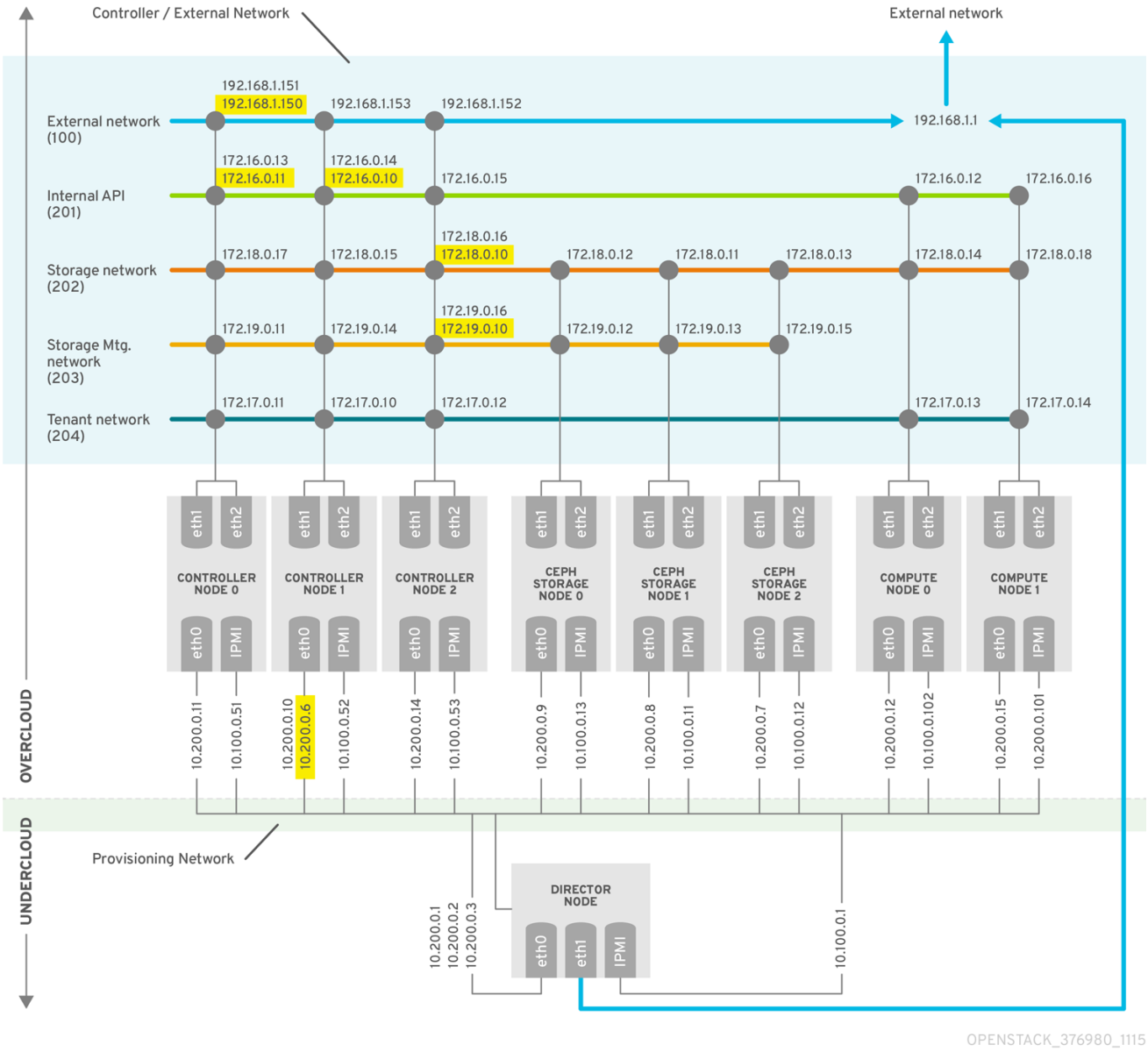
第 2 章 示例部署：使用 COMPUTE 和 CEPH 的高可用性集群

本例场景显示架构、硬件和网络规格，以及 undercloud 和 overcloud 配置文件，用于使用 OpenStack Compute 服务和 Red Hat Ceph Storage 进行高可用性部署。

重要

此部署旨在用作测试环境的参考，在生产环境中不支持。

图 2.1. 高可用性部署架构示例



2.1. 高可用性硬件规格示例

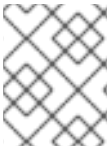
HA 部署示例使用特定的硬件配置。您可以根据您自己的测试部署中的需要调整 CPU、内存、存储或 NIC。

表 2.1. 物理计算机

计算机数量	用途	CPU	内存	磁盘空间	电源管理	NIC
1	undercloud 节点	4	24 GB	40 GB	IPMI	2 (1 外部 1 个；1 置备时 1 个)+ 1 个 IPMI
3	Controller 节点	4	24 GB	40 GB	IPMI	3 （在 overcloud 上 2 绑定；1 on provisioning ）+ 1 IPMI
3	Ceph Storage 节点	4	24 GB	40 GB	IPMI	3 （在 overcloud 上 2 绑定；1 on provisioning ）+ 1 IPMI
2	Compute 节点（根据需要添加更多）	4	24 GB	40 GB	IPMI	3 （在 overcloud 上 2 绑定；1 on provisioning ）+ 1 IPMI

2.2. 高可用性网络规格示例

示例 HA 部署使用特定的虚拟网络和物理网络配置。您可以根据您自己的测试部署中的需要调整配置。



注意

本例不包括 control plane 的硬件冗余以及配置了 overcloud keystone 管理端点的 provisioning 网络。有关规划高可用性网络的详情，请参考 第 1.3 节 “规划高可用性网络”。

表 2.2. 物理和虚拟网络

物理 NIC	用途	vlan	描述
eth0	置备网络(undercloud)	N/A	管理 director (undercloud) 中的所有节点.
eth1 和 eth2	Controller/External (overcloud)	N/A	使用 VLAN 绑定 NIC

物理 NIC	用途	vlan	描述
	外部网络	VLAN 100	允许从环境外部访问项目网络、内部 API 和 OpenStack Horizon 仪表板
	内部 API	VLAN 201	提供对 Compute 节点和 Controller 节点间内部 API 的访问
	存储访问	VLAN 202	将 Compute 节点连接到存储介质
	存储管理	VLAN 203	管理存储介质
	项目网络	VLAN 204	为 RHOSP 提供项目网络服务

2.3. 高可用性 UNDERCLOUD 配置文件示例

示例 HA 部署使用stackenv. **json、 undercloud. conf** 和 **network-environment.yaml** 中的 undercloud 配置文件。

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. 高可用性 OVERCLOUD 配置文件示例

示例 HA 部署使用 overcloud 配置文件 **haproxy.cfg**、**corosync.cfg** 和 **ceph.cfg**。

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

此文件标识 HAProxy 管理的服务。它包含 HAProxy 监控的服务的设置。该文件在所有 Controller 节点上相同。

```
# 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 文件（控制器节点）

此文件定义集群基础架构，所有 Controller 节点上都可用。

```
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 节点)

此文件包含 Ceph 高可用性设置，包括监控主机的主机名和 IP 地址。

```
[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
```

2.5. 其他资源

- 使用容器化 Red Hat Ceph 部署 Overcloud
- 第 1 章 *Red Hat OpenStack Platform 高可用性概述和规划*

第 3 章 使用 PACEMAKER 管理高可用性服务

Pacemaker 服务管理核心容器和主动 - 被动服务，如 Galera、Rabing、Redis 和 HAProxy。您可以使用 Pacemaker 查看和管理有关受管服务、虚拟 IP 地址、电源管理和隔离的一般信息。

3.1. PACEMAKER 资源捆绑包和容器

Pacemaker 将 Red Hat OpenStack Platform (RHOSP)服务作为 Bundle Set 资源或捆绑包进行管理。其中的大部分服务都是以相同方式启动的主动服务，并且始终在每个 Controller 节点上运行。

Pacemaker 管理以下资源类型：

捆绑包（Bundle）

捆绑包资源在所有 Controller 节点上配置和复制同一容器，将必要的存储路径映射到容器目录，并设置与资源本身相关的特定属性。

Container

容器可以运行不同类型的资源，从简单的 **systemd** 服务（如 HAProxy）到复杂的服务（如 Galera），这需要特定的资源代理来控制和设置不同节点上的服务状态。

重要

- 您不能使用 **podman** 或 **systemctl** 管理捆绑包或容器。您可以使用命令检查服务的状态，但您必须使用 Pacemaker 对这些服务执行操作。
- Pacemaker 控制的 podman 容器由 Podman 将 **RestartPolicy** 设置为 **no**。这是为了确保 Pacemaker 而不是 Podman，控制容器启动和停止操作。

3.1.1. 简单捆绑包集资源（简单捆绑包）

一个简单的 Bundle Set 资源或简单捆绑包是一组容器，每个容器都包含要在 Controller 节点上部署的相同 Pacemaker 服务。

以下示例显示了 **pcs status** 命令的输出中的简单捆绑包列表：

```
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
```

对于每个捆绑包，您可以查看以下详情：

- Pacemaker 为服务分配的名称。
- 对与捆绑包关联的容器的引用。
- 在不同 Controller 节点上运行的副本列表和状态。

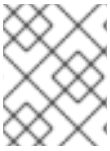
以下示例显示了 **haproxy-bundle** 简单捆绑包的设置：

```
$ 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)
```

示例显示有关捆绑包中容器的以下信息：

- **镜像** ：容器使用的镜像，引用 undercloud 的本地 registry。
- **网络** ：容器网络类型，本例中为 **"host** "。
- **选项** ：容器的特定选项。
- **副本** ：指示集群中必须运行多少个容器副本。每个捆绑包都包含三个容器，每个 Controller 节点对应一个容器。
- **run-command** ：用于生成容器的系统命令。
- **存储映射** ：将每个主机上的本地路径映射到容器。**haproxy** 配置位于 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 文件中，而不是 **/etc/haproxy/haproxy.cfg** 文件中。



注意

虽然 HAProxy 通过对所选服务进行负载均衡流量提供高可用性服务，但您可以通过将其配置为 Pacemaker 捆绑包服务，将 HAProxy 配置为高度可用的服务。

3.1.2. 复杂的捆绑包集资源（复杂捆绑包）

复杂的 Bundle Set 资源或复杂捆绑包是 Pacemaker 服务，除了包含在简单捆绑包中的基本容器配置外，还要指定资源配置。

此配置需要管理多状态资源，这些资源是不同状态的服务，具体取决于它们运行的 Controller 节点。

这个示例显示 **pcs status** 命令的输出中的复杂捆绑包列表：

```
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
```

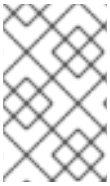
此输出显示了每个复杂捆绑包的以下信息：

- RabbitMQ ：所有三个 Controller 节点都运行服务的独立实例，类似于一个简单的捆绑包。
- Galera：在相同限制下，所有三个 Controller 节点都作为 Galera master 运行。
- redis： **overcloud-controller-0** 容器作为 master 运行，其他两个 Controller 节点则作为从设备运行。每个容器类型可能在不同约束下运行。

以下示例显示了 **galera-bundle** 复杂捆绑包的设置：

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

此输出显示，这与简单的捆绑包不同，**galera-bundle** 资源包含显式资源配置，用于决定多状态资源的所有方面。



注意

虽然服务可以同时在多个 Controller 节点上运行，但 Controller 节点本身可能无法侦听访问这些服务所需的 IP 地址。有关如何检查服务的 IP 地址的详情，请参考 第 3.4 节 “查看高可用性集群中虚拟 IP 的资源信息”。

3.2. 检查 PACEMAKER 集群状态

您可以在 Pacemaker 运行的任何节点中检查 Pacemaker 集群的状态，并查看活跃和运行的资源数量的信息。

先决条件

- 高可用性已部署并运行。

流程

1. 以 **heat-admin** 用户身份登录任何 Controller 节点。

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

2. 运行 **pcs status** 命令：

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

输出示例：

```
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:
[...]
```

输出的主要部分显示有关集群的以下信息：

- **Cluster name** ：集群的名称。
- **[NUM] 节点配置** ：为集群配置的节点数量。
- **[NUM]** 资源配置 : 为集群配置的资源数量。

- **在线** ：当前在线的 Controller 节点的名称。
- **GuestOnline** ：当前在线的客户机节点的名称。每个客户机节点包含一个复杂的 Bundle Set 资源。有关捆绑包集的更多信息，请参阅 第 3.1 节 “Pacemaker 资源捆绑包和容器”。

3.3. 检查高可用性集群中的捆绑包状态

您可以从 undercloud 节点检查捆绑包的状态，或登录到其中一个 Controller 节点，以直接检查捆绑包状态。

先决条件

- 高可用性已部署并运行。

流程

使用以下选项之一：

- 登录到 undercloud 节点并检查捆绑包状态，在本例中为 **haproxy-bundle** ：

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

输出示例：

```
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
```

输出显示 **haproxy** 进程在容器内运行。

- 登录到 Controller 节点并检查捆绑包状态，在本例中为 **haproxy** ：

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

输出示例：

```
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. 查看高可用性集群中虚拟 IP 的资源信息

要检查所有虚拟 IP (VIP)或特定 VIP 的状态，请使用相关选项运行 **pcs resource show** 命令。每个 IPaddr2 资源都设置客户端用来请求对服务访问的虚拟 IP 地址。如果具有该 IP 地址的 Controller 节点失败，IPaddr2 资源会将 IP 地址重新分配给不同的 Controller 节点。

先决条件

- 高可用性已部署并运行。

流程

1. 以 **heat-admin** 用户身份登录任何 Controller 节点。

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

2. 使用以下选项之一：

- 使用 **--full** 选项运行 **pcs resource show** 命令显示使用虚拟 IP 的所有资源：

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

输出示例：

```
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
```

每个 IP 地址最初都附加到特定的 Controller 节点。例如，192. **168**.150 在 **overcloud-controller-0** 上启动。但是，如果该 Controller 节点失败，IP 地址会被重新分配给集群中的其他 Controller 节点。

下表描述了示例输出中的 IP 地址，并显示每个 IP 地址的原始分配。

表 3.1. IP 地址描述和分配源

IP 地址	描述	从中分配
10.200.0.6	控制器虚拟 IP 地址	在 **undercloud.conf** 文件中，**dhcp_start** 和 **dhcp_end** 范围的一部分设置为 **10.200.0.5-10.200.0.24**
192.168.1.150	公共 IP 地址	**network-environment.yaml** 文件中的 **ExternalAllocationPools** 属性
172.16.0.10	提供对 Controller 节点上的 OpenStack API 服务的访问	**network-environment.yaml** 文件中的 **InternalApiAllocationPools**

IP 地址	描述	从中分配
172.16.0.11	提供对 Controller 节点上的 Redis 服务的访问	**network-environment.yaml** 文件中的 **InternalApiAllocationPools**
172.18.0.10	提供对 Glance API 和 Swift 代理服务访问权限的存储虚拟 IP 地址	**network-environment.yaml** 文件中的 **StorageAllocationPools** 属性
172.19.0.10	提供对存储管理的访问	**network-environment.yaml** 文件中的 **StorageMgmtAlloctionPools**

- 使用使用该 VIP 的资源名称运行 **pcs resource show** 命令来查看特定的 VIP 地址，在本例中为 **ip-192.168.1.150** ：

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

输出示例：

```
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. 查看高可用性集群中的虚拟 IP 的网络信息

您可以查看分配给特定虚拟 IP (VIP)的 Controller 节点的网络接口信息，并查看特定服务的端口号分配。

先决条件

- 高可用性已部署并运行。

流程

1. 登录到分配给您要查看的 IP 地址的 Controller 节点，并在网络接口上运行 **ip addr show** 命令，在本例中为 **vlan100** ：

```
$ ip addr show vlan100
```

输出示例：

```
 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. 运行 **netstat** 命令显示侦听 IP 地址的所有进程，在本例中为 **192.168.1.150.haproxy** ：

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

输出示例：

```
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
```

注意

侦听所有本地地址的进程（如 **0.0.0.0** ）也可以通过 **192.168.1.150** 获得。这些进程包括 **sshd**、**mysqld**、**dhclient**、**ntpd**。

3. 通过打开 HA 服务的配置文件来查看它们侦听的默认端口号分配，在本例中为 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ：

- TCP 端口 6080: **nova_novncproxy**
- TCP 端口 9696: **neutron**
- TCP 端口 8000: **heat_cfn**
- TCP 端口 80: **horizon**
- TCP 端口 8776: **cinder**
  在本例中，**haproxy.cfg** 文件中定义的大多数服务侦听所有三个 Controller 节点上的 **192.168.1.150** IP 地址。但是，只有 **controller-0** 节点正在从外部侦听 **192.168.1.150** IP 地址。

  因此，如果 **controller-0** 节点失败，HAProxy 只需要将 **192.168.1.150** 重新分配给另一个 Controller 节点，所有其他服务都将已在回退 Controller 节点上运行。

3.6. 检查隔离代理和 PACEMAKER 守护进程状态

您可以检查隔离代理的状态，以及 Pacemaker 运行的任何节点中 Pacemaker 守护进程的状态，并查看活跃和运行的 Controller 节点数量的信息。

先决条件

- 高可用性已部署并运行。

流程

1. 以 **heat-admin** 用户身份登录任何 Controller 节点。

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

2. 运行 **pcs status** 命令：

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

输出示例：

```
 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
```

输出显示 **pcs status** 命令的输出的以下部分：

- **my-ipmilan-for-controller** ：显示每个 Controller 节点的隔离类型 (**stonith:fence_ipmilan**)，以及 IPMI 服务是否停止或正在运行。
- **PCSD 状态** ：显示所有三个 Controller 节点当前在线。
- **守护进程状态** ：包含三个 Pacemaker 守护进程的状态： **corosync**、**pacemaker** 和 **pcsd**。在示例中，所有三个服务都处于活动状态并启用。

3.7. 其他资源

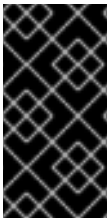
- 配置和管理高可用性集群

第 4 章 使用 STONITH 的隔离 CONTROLLER 节点

隔离(fencing)是隔离故障节点来保护集群和集群资源的过程。如果没有隔离，故障节点可能会导致集群中的数据崩溃。director 使用 Pacemaker 提供 Controller 节点的高可用性集群。

Pacemaker 使用名为 STONITH 的进程来隔离失败的节点。STONITH 是 "Shoot the other nodes in the head" 的缩写。STONITH 默认禁用，需要手动配置，以便 Pacemaker 能够控制集群中每个节点的电源管理。

如果 Controller 节点失败健康检查，则作为 Pacemaker 指定的协调器(DC)的 Controller 节点使用 Pacemaker **stonith** 服务来隔离受影响的 Controller 节点。



重要

不支持在不使用 STONITH 的情况下部署高可用性 overcloud。您必须在高可用性 overcloud 中为作为 Pacemaker 集群一部分的每个节点配置 STONITH 设备。有关 STONITH 和 Pacemaker 的信息，请参阅红帽高可用性集群中的隔离和 RHEL 高可用性集群的支持策略。

4.1. 支持的隔离代理

当您使用隔离部署高可用性环境时，您可以根据环境需要选择隔离代理。要更改隔离代理，您必须在 fence **.yaml** 文件中配置附加参数。

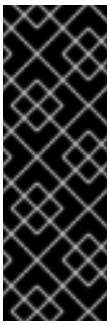
Red Hat OpenStack Platform (RHOSP)支持以下隔离代理：

智能平台管理接口 (IPMI)

Red Hat OpenStack Platform (RHOSP)用来管理隔离的默认隔离机制。

STONITH 块设备(SBD)

SBD （基于存储）守护进程与 Pacemaker 和 watchdog 设备集成，以便在触发隔离机制时调度节点以可靠地关闭，并在传统隔离机制不可用时可靠地关闭。



重要

- 使用 **pacemaker_remote** 的远程裸机或虚拟机节点不支持 SBD 隔离，因此如果部署使用 Instance HA，则不支持它。
- 不支持使用块存储设备的 **fence_sbd** 和 **sbd poison-pill** 隔离。
- SBD 隔离只支持兼容 watchdog 设备。如需更多信息，请参阅 RHEL 高可用性集群的支持政策 - sbd 和 fence_sbd。

fence_kdump

在带有 **kdump** 崩溃恢复服务的部署中使用。如果您选择此代理，请确保您有足够的磁盘空间来存储转储文件。

除了 IPMI、**fence_ the 或 Redfish 隔离代理外，您还可以将此代理** 配置为辅助机制。如果您配置多个隔离代理，请确保为第一个代理分配足够时间来完成任务，然后再启动下一个任务。

重要

- RHOSP director 仅支持 **fence_kdump** STONITH 代理的配置，不支持隔离代理依赖的完整 **kdump** 服务的配置。有关配置 **kdump** 服务的详情，请参考红帽知识库解决方案 如何在 Red Hat Pacemaker 集群中配置 fence_kdump 。
- 如果 Pacemaker 网络流量接口使用 **ovs_bridges**、**ovs_bonds** 或 Linux 网桥上的 VLAN，则不支持 **fence_kdump**。要启用 **fence_kdump**，您必须将网络设备改为 **linux_bond** 或 **linux_bridge**。有关 VLAN 和 **kdump** 的详情，请查看红帽知识库解决方案 kdump 支持哪些 VLAN 配置？有关网络接口配置的更多信息，请参阅 网络接口参考。

Redfish

在部署中使用支持 DMTF Redfish API 的服务器。要指定此代理，请在 fence **.yaml** 文件中将 **agent** 参数的值改为 **fence_redfish**。有关 Redfish 的更多信息，请参阅 DTMF 文档。

fence_ASP for Red Hat Virtualization (RHV)

用于为在 RHV 环境中运行的 Controller 节点配置隔离。您可以像 IPMI **隔离一样生成 fence.yaml** 文件，但您必须在 **nodes.json** 文件中定义 **pm_type** 参数以使用 RHV。
默认情况下，**ssl_insecure** 参数设置为接受自签名证书。您可以根据安全要求更改参数值。

重要

确保您使用在 RHV 中创建和启动虚拟机的角色，如 **UserVMManager**。

多层隔离

您可以配置多个隔离代理来支持复杂的隔离用例。例如，您可以将 IPMI 隔离配置为 **fence_kdump**。隔离代理的顺序决定了 Pacemaker 触发每个机制的顺序。

其他资源

- 第 4.2 节 “在 overcloud 上部署隔离”
- 第 4.3 节 “在 overcloud 上测试隔离”
- 第 4.5 节 “隔离参数”

4.2. 在 OVERCLOUD 上部署隔离

要在 overcloud 上部署隔离，首先查看 STONITH 和 Pacemaker 的状态并配置 **fence.yaml** 文件。然后，部署 overcloud 并配置其他参数。最后，测试是否在 overcloud 上正确部署隔离。

先决条件

- 为您的部署选择正确的隔离代理。有关支持的隔离代理列表，请参阅 第 4.1 节 “支持的隔离代理”。
- 确保您可以访问 director 中注册节点时创建的 **nodes.json** 文件。此文件是您在部署期间生成的 **fence.yaml** 文件所需的输入。
- **nodes.json** 文件必须包含节点上其中一个网络接口(NIC)的 MAC 地址。有关更多信息，请参阅为 Overcloud 注册节点。

- 如果您使用 Red Hat Virtualization (RHV)隔离代理，请使用具有管理虚拟机权限的角色，如 **UserVMManager**。

流程

1. 以 **heat-admin** 用户身份登录每个 Controller 节点。
2. 验证集群是否正在运行：

```
$ sudo pcs status
```

输出示例：

```
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. 验证 STONITH 是否已禁用：

```
$ sudo pcs property show
```

输出示例：

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

4. 根据您要使用的隔离代理，选择以下选项之一：
   - 如果使用 IPMI 或 RHV 隔离代理，请生成 **fence.yaml** 环境文件：

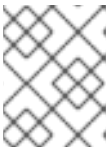
```
(undercloud) $ openstack overcloud generate fencing --output fencing.yaml nodes.json
```

注意

此命令将 **ilo** 和 **drac** 电源管理详情转换为 IPMI 等效。

   - 如果您使用不同的隔离代理，如 STONITH 块设备(SBD)、**fence_kdump** 或 Redfish，或者使用预置备节点，请手动创建 **fence.yaml** 文件。
5. 仅限 SBD 隔离 ：在 **fence.yaml** 文件中添加以下参数：

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



注意

此步骤仅适用于初始 overcloud 部署。有关如何在现有 overcloud 上启用 SBD 隔离的更多信息，请参阅 在 RHEL 7 和 8 中启用 sbd 隔离 。

6. 仅多层隔离：将特定于 level 的参数添加到生成的 **fence.yaml** 文件中：

```
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>
```

将 **<** parameter& **gt; 和 <value** > 替换为隔离代理所需的实际参数和值。

7. 运行 **overcloud deploy** 命令，并包含 **fence.yaml** 文件以及与部署相关的任何其他环境文件：

```
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. 仅 SBD 隔离：设置 watchdog 计时器设备间隔，并检查间隔是否已正确设置。

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

验证

1. 以 **heat-admin** 用户身份登录 overcloud，并确保 Pacemaker 配置为资源管理器：

```
$ 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
```

在本例中，Pacemaker 配置为对 **fence.yaml** 文件中指定的每个 Controller 节点使用 STONITH 资源。

注意

您不能在它控制的同一节点上配置 **fence-resource** 进程。

2. 检查隔离资源属性。STONITH 属性值必须与 **fence.yaml 文件中的值匹配** ：

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

其它资源

- 第 4.3 节 “在 overcloud 上测试隔离”
- 第 4.5 节 “隔离参数”
- 探索 RHEL 高可用性的组件 - sbd 和 fence_sbd

4.3. 在 OVERCLOUD 上测试隔离

要测试隔离是否正常工作，请通过关闭 Controller 节点上的所有端口并重新启动服务器来触发隔离。

重要

此流程有意丢弃与 Controller 节点的所有连接，这会导致节点重启。

先决条件

- 隔离已在 overcloud 上部署并运行。有关如何部署隔离的详情，请参考 第 4.2 节 “在 overcloud 上部署隔离”。
- Controller 节点可用于重启。

流程

1. 以 **stack** 用户身份登录 Controller 节点，并提供凭据文件：

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

2. 进入 **root** 用户并关闭到 Controller 节点的所有连接：

```
$ 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. 在不同的 Controller 节点上，在 Pacemaker 日志文件中找到隔离事件：

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

如果 STONITH 服务对 Controller 执行隔离操作，则日志文件会显示隔离事件。

4. 等待几分钟，然后通过运行 **pcs status** 命令验证重启的 Controller 节点是否在集群中再次运行。如果您在输出中看到重启的 Controller 节点，隔离功能可以正常工作。

4.4. 查看 STONITH 设备信息

要查看 STONITH 如何配置隔离设备，请从 overcloud 运行 **pcs stonith show --full** 命令。

先决条件

- 隔离已在 overcloud 上部署并运行。有关如何部署隔离的详情，请参考 第 4.2 节 “在 overcloud 上部署隔离”。

流程

- 显示 Controller 节点列表及其 STONITH 设备的状态：

```
$ 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)
```

此输出显示每个资源的以下信息：

  - 隔离设备用来根据需要打开和关闭机器的 IPMI 电源管理服务，如 **fence_ipmilan**。
  - IPMI 接口的 IP 地址，如 **10.100.0.51**。
  - 要使用登录的用户名，如 **admin**。
  - 用于登录节点的密码，如 **abc**。
  - 监控每个主机的时间间隔（以秒为单位），如 **60s**。

4.5. 隔离参数

当您在 overcloud 上部署隔离时，您可以使用必要的参数生成 **fence.yaml** 文件来配置隔离。

以下示例显示了 **fence.yaml** 环境文件的结构：

```
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
```

此文件包含以下参数：

EnableFencing

为 Pacemaker 管理的节点启用隔离功能。

FencingConfig

列出隔离设备以及每个设备的参数：

- **代理** ：隔离代理名称。
- **host_mac** ：调配接口的 mac 地址或服务器上的任何其他网络接口。您可以使用它作为隔离设备的唯一标识符。
- **参数** ：隔离设备参数列表。

隔离设备参数

列出隔离设备参数。本例显示 IPMI 隔离代理的参数：

- **Auth** ：IPMI 身份验证类型(**md5**、 **密码**、或 none)。
- **IPaddr** ：IPMI IP 地址。
- **ipport** ：IPMI 端口。
- **登录** ： IPMI 设备的用户名。
- **passwd** ：IPMI 设备的密码。
- **lanplus**: 使用 lanplus 提高连接安全性。
- **privlvl**: IPMI 设备的权限级别
- **pcmk_host_list** ：Pacemaker 主机列表。

其他资源

- 第 4.2 节 “在 overcloud 上部署隔离”
- 第 4.1 节 “支持的隔离代理”

4.6. 其他资源

- “在 Red Hat High Availability 集群中配置隔离功能”

第 5 章 使用 HAPROXY 进行负载均衡流量

HAProxy 服务为高可用性集群中的 Controller 节点提供流量负载均衡，以及日志记录和示例配置。**haproxy** 软件包包含 **haproxy** 守护进程，它对应于同一名称的 **systemd** 服务。Pacemaker 将 HAProxy 服务作为一个名为 **haproxy-bundle** 的高可用服务进行管理。

5.1. HAPROXY 如何工作

director 可以配置大多数 Red Hat OpenStack Platform 服务以使用 HAProxy 服务。director 在 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 文件中配置这些服务，该文件指示 HAProxy 在每个 overcloud 节点上运行专用容器中运行。

下表显示了 HAProxy 管理的服务列表：

表 5.1. 由 HAProxy 管理的服务

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

对于 **haproxy.cfg** 文件中的每个服务，您可以看到以下属性：

- **listen** ：侦听请求的服务名称。
- **bind** ：服务正在侦听的 IP 地址和 TCP 端口号。
- **服务器** ：使用 HAProxy、IP 地址和侦听端口的每个 Controller 节点服务器的名称，以及服务器的附加信息。

以下示例显示了 **haproxy.cfg** 文件中的 OpenStack Block Storage (cinder)服务配置：

```
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
```

本例输出显示了有关 OpenStack Block Storage (cinder)服务的以下信息：

- **172.16.0.10:8776**: 在 overcloud 中使用的 Internal API 网络 (VLAN201) 的虚拟 IP 地址和端口。
- **192.168.1.150:8776** ：外部网络(VLAN100)上的虚拟 IP 地址和端口，从 overcloud 外部访问 API 网络。

- **8776**: OpenStack Block Storage (cinder)服务正在侦听的端口号。
- **服务器** ： Controller 节点名称和 IP 地址。HAProxy 可以将对这些 IP 地址发出的请求定向到 **服务器** 输出中列出的一个 Controller 节点。
- **httpchk** ：在 Controller 节点服务器上启用健康检查。
- **回退 5** ：失败的健康检查数量，以确定该服务离线。
- **between 2000** ：以毫秒为单位在两个连续健康检查之间间隔。
- **最后** 2：用于确定该服务是否正在运行的成功健康检查数量。

有关您可以在 **haproxy.cfg** 文件中使用的设置的更多信息，请参阅安装 **haproxy** 软件包的任何节点上的 **/usr/share/doc/haproxy-[VERSION]/configuration.txt** 文件。

5.2. 查看 HAPROXY STATS

默认情况下，director 也在所有 HA 部署中启用 HAProxy Stats 或 statistics。使用此功能，您可以在 HAProxy Stats 页面中查看有关数据传输、连接和服务器状态的详细信息。

director 还设置您用来访问 HAProxy Stats 页面的 **IP:Port** 地址，并将信息存储在 **haproxy.cfg** 文件中。

先决条件

- 高可用性已部署并运行。

流程

1. 在安装了 HAProxy 的任何 Controller 节点上打开 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 文件。
2. 找到 **listen haproxy.stats** 部分：

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

3. 在 Web 浏览器中，导航到 **10.200.0.6:1993**，并从 **stats auth** 行输入凭据来查看 HAProxy Stats 页面。

5.3. 其他资源

- HAProxy 1.8 文档
- 如何验证 my haproxy.cfg 是否已正确配置为负载均衡 openstack 服务？

第 6 章 使用 GALERA 管理数据库复制

Red Hat OpenStack Platform 使用 MariaDB Galera 集群来管理数据库复制。Pacemaker 将 Galera 服务作为捆绑包集资源运行，用于管理数据库 master/slave 状态。您可以使用 Galera 测试和验证数据库集群的不同方面，如主机名解析、集群完整性、节点完整性和数据库复制性能。

当您调查数据库集群的完整性时，每个节点都必须满足以下条件：

- 节点是正确的集群的一部分。
- 节点可以写入集群。
- 节点可以从集群接收查询和写入命令。
- 节点连接到集群中的其他节点。
- 节点将写集复制到本地数据库中的表。

6.1. 在 MARIADB 集群中验证主机名解析

要对 MariaDB Galera 集群进行故障排除，首先删除所有主机名解析问题，然后检查每个 Controller 节点数据库上的 write-set 复制状态。要访问 MySQL 数据库，请在 overcloud 部署期间使用 director 设置的密码。

默认情况下，director 将 Galera 资源绑定到主机名而不是 IP 地址。因此，防止主机名解析（如错误配置或 DNS）的任何问题可能会导致 Pacemaker 错误地管理 Galera 资源。

流程

1. 从 Controller 节点，运行 **hiera** 命令获取 MariaDB 数据库 root 密码。

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

2. 获取节点上运行的 MariaDB 容器的名称。

```
$ 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. 从每个节点上的 MariaDB 数据库获取 write-set 复制信息。

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

每个相关变量都使用前缀 **wsrep**。

4. 通过检查集群是否报告正确的节点数量，来验证 MariaDB Galera 集群的健康和完整性。

6.2. 检查 MARIADB 集群的完整性

要调查 MariaDB Galera 集群的问题，请通过检查每个 Controller 节点上的特定 **wsrep** 数据库变量来检查整个集群的完整性。

流程

- 运行以下命令，并将 &lt **;variable** > 替换为您要检查的 **wsrep** 数据库变量：

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

以下示例演示了如何查看节点的集群状态 UUID：

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

下表列出了可用于检查集群完整性的 **wsrep** 数据库变量。

表 6.1. 用于检查集群完整性的数据库变量

变量	概述	描述
wsrep_cluster_state_uuid	集群状态 UUID	节点所属的集群的 ID。所有节点必须具有相同的集群 ID。具有不同 ID 的节点没有连接到集群。
wsrep_cluster_size	集群中的节点数	您可以在任何节点上检查它。如果值小于实际的节点数量，则某些节点会失败或丢失连接。
wsrep_cluster_conf_id	集群更改总数	决定集群是否被分成几个组件，还是分区。分区通常由网络失败造成的。所有节点必须具有相同的值。 如果某些节点报告不同的 **wsrep_cluster_conf_id**，请检查 **wsrep_cluster_status** 值，以查看节点是否仍然可以写入集群 (**主**)。

变量	概述	描述
wsrep_cluster_status	主要组件状态	确定节点是否可以写入集群。如果节点可以写入集群，则 **wsrep_cluster_status** 值为 **Primary**。任何其他值表示节点是不可调度的分区的一部分。

6.3. 检查 MARIADB 集群中的数据库节点完整性

要调查 MariaDB Galera 集群中特定 Controller 节点的问题，请检查特定的 **wsrep** 数据库变量来检查节点的完整性。

流程

- 运行以下命令，并将 &lt **;variable** > 替换为您要检查的 **wsrep** 数据库变量：

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

下表列出了可用于检查节点完整性的 **wsrep** 数据库变量。

表 6.2. 用于检查节点完整性的数据库变量

变量	概述	描述
wsrep_ready	接受查询的节点功能	说明节点是否可以接受来自集群的 write-sets。如果是这样，则 **wsrep_ready** 为 **ON**。
wsrep_connected	节点网络连接	说明节点是否可以连接到网络上的其他节点。如果是这样，则 **wsrep_connected** 为 **ON**。

变量	概述	描述
wsrep_local_state_comment	节点状态	总结节点状态。如果节点可以写入集群，则 **wsrep_local_state_comment** 的典型值可以加入、**waiting on SST**、joined、**synced** 或 **Donor**。 **如果节点是无法正常工作组件的一部分，则 wsrep_local_state_comment 的值为 Initialized。**

注意

- 即使节点只连接到集群中某个节点的子集，**wsrep_connected** 值可能是 **ON**。例如，对于集群分区，节点可能是无法写入集群的组件的一部分。有关检查集群完整性的更多信息，请参阅 第 6.2 节 “检查 MariaDB 集群的完整性”。
- 如果 **wsrep_connected** 值为 **OFF**，则该节点不会连接到任何集群组件。

6.4. 在 MARIADB 集群中测试数据库复制性能

要检查 MariaDB Galera 集群的性能，请通过检查特定的 **wsrep** 数据库变量，对集群的复制吞吐量运行基准测试测试。

每次查询其中一个变量时，**FLUSH STATUS** 命令会重置变量值。要运行基准测试测试，您必须运行多个查询并分析变体。这些变体可帮助您确定影响集群性能的流控制量。

流控制是集群用来管理复制的机制。当本地接收队列超过特定阈值时，Flow Control 会暂停复制，直到队列大小停机为止。有关流控制的更多信息，请参阅 Galera 集群网站上的流控制。

流程

- 运行以下命令，并将 &lt **;variable** > 替换为您要检查的 **wsrep** 数据库变量：

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

下表列出了可用于测试数据库复制性能的 **wsrep** 数据库变量。

表 6.3. 用于检查数据库复制性能的数据库变量

变量	概述	使用方法
wsrep_local_recv_queue_avg	最后一次查询后本地收到的写设置队列的平均大小。	高于 0.0 的值表示节点无法像收到写入集一样快速应用 write-sets，这会触发复制节流。检查 **wsrep_local_recv_queue_min** 和 **wsrep_local_recv_queue_max** 查看此基准。
wsrep_local_send_queue_avg	最后一次查询后平均发送队列长度。	高于 0.0 的值表示复制节流和网络吞吐量问题的可能性较高。
wsrep_local_recv_queue_min and **wsrep_local_recv_queue_max**	最后一次查询后本地接收队列的最小和最大大小。	如果 **wsrep_local_recv_queue_avg** 的值高于 0.0，您可以检查这些变量以确定队列大小的范围。
wsrep_flow_control_paused	流控制在最后一次查询后暂停节点的时间分数。	高于 0.0 的值表示流控制暂停节点。要确定暂停的持续时间，请将 **wsrep_flow_control_paused** 值乘以查询间隔的秒数。最佳值尽可能接近 0.0。 例如： ● 如果在最后一次查询的 1 分钟后 **wsrep_flow_control_paused** 的值为 0.50，则 Flow Control 会暂停节点 30 秒。 ● 如果在最后一次查询后 **wsrep_flow_control_paused** 的值为 1.0，则流控制会在整个分钟内暂停节点。
wsrep_cert_deps_distance	并行应用的最低和最高序列号 (**seqno**)值之间的平均区别	如果节流和暂停，此变量表示可以并行应用平均多少写集。将值与 **wsrep_slave_threads** 变量进行比较，以查看实际可同时应用多少写集。

变量	概述	使用方法
wsrep_slave_threads	可同时应用的线程数量	您可以增加此变量的值来同时应用更多线程，这将增加 **wsrep_cert_deps_distance** 的值。**wsrep_slave_threads** 的值不得高于节点上的 CPU 内核数。 例如，如果 **wsrep_cert_deps_distance** 值为 **20**，您可以将 **wsrep_slave_threads** 的值从 **2** 增加到 **4**，以增加节点可以应用的写集量。 如果有问题的节点已有最佳的 **wsrep_slave_threads** 值，您可以在调查可能的连接问题时从集群中删除该节点。

6.5. 其他资源

- 什么是 MariaDB Galera 集群？

第 7 章 对高可用性资源进行故障排除

如果资源失败，您必须调查问题的原因和位置，修复失败的资源，并选择性地清理资源。根据部署，资源失败有很多可能的原因，您必须调查资源以确定如何解决这个问题。

例如，您可以检查资源限制，以确保资源不会相互中断，并且资源可以相互连接。您还可以检查比其他 Controller 节点频繁隔离的 Controller 节点，以识别可能的通信问题。

根据资源问题的位置，您可以选择以下选项之一：

Controller 节点问题

如果对 Controller 节点的健康检查失败，这可能表示 Controller 节点之间的通信问题。要调查，请登录 Controller 节点，再检查服务是否已正确启动。

单个资源问题

如果控制器上的大多数服务都正确运行，您可以运行 **pcs status** 命令并检查输出有关特定 Pacemaner 资源失败的信息，或运行 **systemctl** 命令来调查非 Pacemaker 资源失败。

7.1. 查看高可用性集群中的资源限制

在调查资源问题前，您可以查看服务启动方式的限制，包括与每个资源相关的限制、资源启动的顺序，以及资源是否必须与另一个资源在一起。

流程

- 使用以下选项之一：
  - 要查看所有资源约束，请登录到任何 Controller 节点并运行 **pcs constraint show** 命令：

```
$ sudo pcs constraint show
```

以下示例显示了 Controller 节点上的 **pcs constraint show** 命令的输出：

```
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)
```

此输出显示以下主要约束类型：

位置约束

列出可为其分配资源的位置：

- 第一个约束定义了一个规则，将 **galera-bundle** 资源设置为在将 **galera-role** 属性设置为 **true** 的节点上运行。
- 第二个位置约束指定 IP 资源 **ip-192.168.24.15** 仅在将 **haproxy-role** 属性设置为 **true** 的节点上运行。这意味着集群将 IP 地址与 **haproxy** 服务相关联，这是使服务可以被访问所必需的。
- 第三个位置约束显示 **ipmilan** 资源在每个 Controller 节点上被禁用。

排序约束

列出资源可以启动的顺序。本例演示了一个约束，它会在 HAProxy 服务之前将虚拟 IP 地址资源 **IPaddr2** 设置为 start。

注意

排序约束只适用于 IP 地址资源和 HAproxy。systemd 管理所有其他资源，因为计算等服务应该中断依赖服务，如 Galera。

共处约束

列出哪些资源必须放在一起。所有虚拟 IP 地址都链接到 **haproxy-bundle** 资源。

- 要查看特定资源的限制，请登录到任何 Controller 节点并运行 **pcs property show** 命令：

```
$ sudo pcs property show
```

输出示例：

```
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
```

在这个输出中，您可以验证资源限制是否已正确设置。例如，所有 Controller 节点的 **galera-role** 属性都是 **true**，这意味着 **galera-bundle** 资源仅在这些节点上运行。

7.2. 调查 PACEMAKER 资源问题

要调查 Pacemaker 管理失败的资源，登录到资源失败的 Controller 节点，并检查资源的状态和日志事件。例如，调查 **openstack-cinder-volume** 资源的状态和日志事件。

先决条件

- 带有 Pacemaker 服务的 Controller 节点
- 查看日志事件的 root 用户权限

流程

1. 登录到资源失败的 Controller 节点。
2. 使用 **grep** 选项运行 **pcs status** 命令以获取服务的状态：

```
# 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. 查看资源的日志事件：

```
# 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. 根据输出和日志中的信息修正失败的资源。

5. 运行 **pcs resource cleanup** 命令来重置资源的状态和故障数。

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

7.3. 调查 SYSTEMD 资源问题

要调查 systemd 管理失败的资源，请登录到资源失败的 Controller 节点，并检查资源的状态和日志事件。例如，调查 **tripleo_nova_conductor** 资源的状态和日志事件。

先决条件

- 带有 systemd 服务的 Controller 节点
- 查看日志事件的 root 用户权限

流程

1. 运行 **systemctl status** 命令显示资源状态和最近的日志事件：

```
[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
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. 查看资源的日志事件：

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

3. 根据输出和日志中的信息修正失败的资源。

4. 重启资源并检查服务的状态：

```
# 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.
```

第 8 章 监控高可用性 RED HAT CEPH STORAGE 集群

当您使用 Red Hat Ceph Storage 部署 overcloud 时，Red Hat OpenStack Platform 使用 **ceph-mon** 监控守护进程来管理 Ceph 集群。director 在所有 Controller 节点上部署守护进程。

8.1. 检查 RED HAT CEPH 监控服务状态

要检查 Red Hat Ceph Storage 监控服务的状态，请登录 Controller 节点并运行 **ceph status** 命令。

流程

- 登录到 Controller 节点，检查 Ceph 监控服务是否正在运行：

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

8.2. 检查 RED HAT CEPH 监控配置

要检查 Red Hat Ceph Storage 监控服务的配置，请登录 Controller 节点或 Red Hat Ceph 节点，并打开 **/etc/ceph/ceph.conf** 文件。

流程

- 登录到 Controller 节点或 Ceph 节点上，再打开 **/etc/ceph/ceph.conf** 文件以查看监控配置参数：

```
[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
```

这个示例显示以下信息：

  - 所有这三个 Controller 节点都配置为使用 **mon_initial_members** 参数来监控 Red Hat Ceph Storage 集群。
  - **172.19.0.11/24** 网络被配置为提供 Controller 节点和 Red Hat Ceph Storage 节点之间的通信路径。
  - Red Hat Ceph Storage 节点被分配给与 Controller 节点独立的网络，监控 Controller 节点的 IP 地址为 **172.18.0.15**、**172.18.0.16** 和 **172.18.0.17**。

8.3. 检查 RED HAT CEPH 节点状态

要检查特定 Red Hat Ceph Storage 节点的状态，请登录该节点并运行 **ceph -s** 命令。

流程

- 登录到 Ceph 节点并运行 **ceph -s** 命令：

```
# 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
```

此示例输出显示 **health** 参数值为 **HEALTH_OK**，这表示 Ceph 节点处于活动状态且健康。输出显示三个 Ceph 监控服务，这些服务在三个 **overcloud-controller** 节点上运行，以及服务的 IP 地址和端口。

8.4. 其他资源

- Red Hat Ceph 产品页面