Red Hat OpenStack Platform 13

高可用性部署和使用情况

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

Last Updated: 2023-03-29

Red Hat OpenStack Platform 13 高可用性部署和使用情况

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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

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.

目录

前言

第 1 章 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。

服务类型

核心容器

核心服务是 *Galera*、*RabbitMQ*、*Redis* 和 *HAProxy*。这些服务在所有 Controller 节点上运行，需要特定的管理和限制启动、停止和重启操作。您可以使用 Pacemaker 启动、管理和故障排除核心容器服务。

注意

RHOSP 使用 MariaDB Galera 集群来管理数据库复制。

Active-passive

主动 - 被动服务一次在一个 Controller 节点上运行，并包含 **openstack-cinder-volume** 等服务。要移动主动 - 被动服务，您必须使用 Pacemaker 以确保正确的 stop-start 序列如下所示。

systemd 和普通容器

systemd 和纯容器服务是可以中断服务的独立服务。因此，如果您重启 Galera 等高可用性服务，则不需要手动重启任何其他服务，如 **nova-api**。您可以使用 systemd 或 Docker 直接管理 systemd 和普通容器服务。

在使用 director 编排 HA 部署时，director 将使用模板和 Puppet 模块来确保所有服务都正确配置和启动。另外，当对 HA 问题进行故障排除时，您必须使用 **docker** 命令或 **systemctl** 命令与 HA 框架中的服务交互。

服务模式

HA 服务可在以下模式之一中运行：

- **active-active** ：Pacemaker 在多个 Controller 节点上运行相同的服务，并使用 HAProxy 在节点间分配流量或使用单个 IP 地址发送到特定的控制器。在某些情况下，HAProxy 使用 Round Robin 调度将流量分发到主动服务。您可以添加更多 Controller 节点来提高性能。
- **主动 - 被动** ：无法以主动-主动模式运行的服务必须在主动 - 被动模式下运行。在这个模式中，一次只有一个服务实例处于活跃状态。例如，HAProxy 使用 stick-table 选项将传入的 Galera 数据库连接请求定向到单个后端服务。这有助于防止从多个 Galera 节点同时连接到同一数据太多。

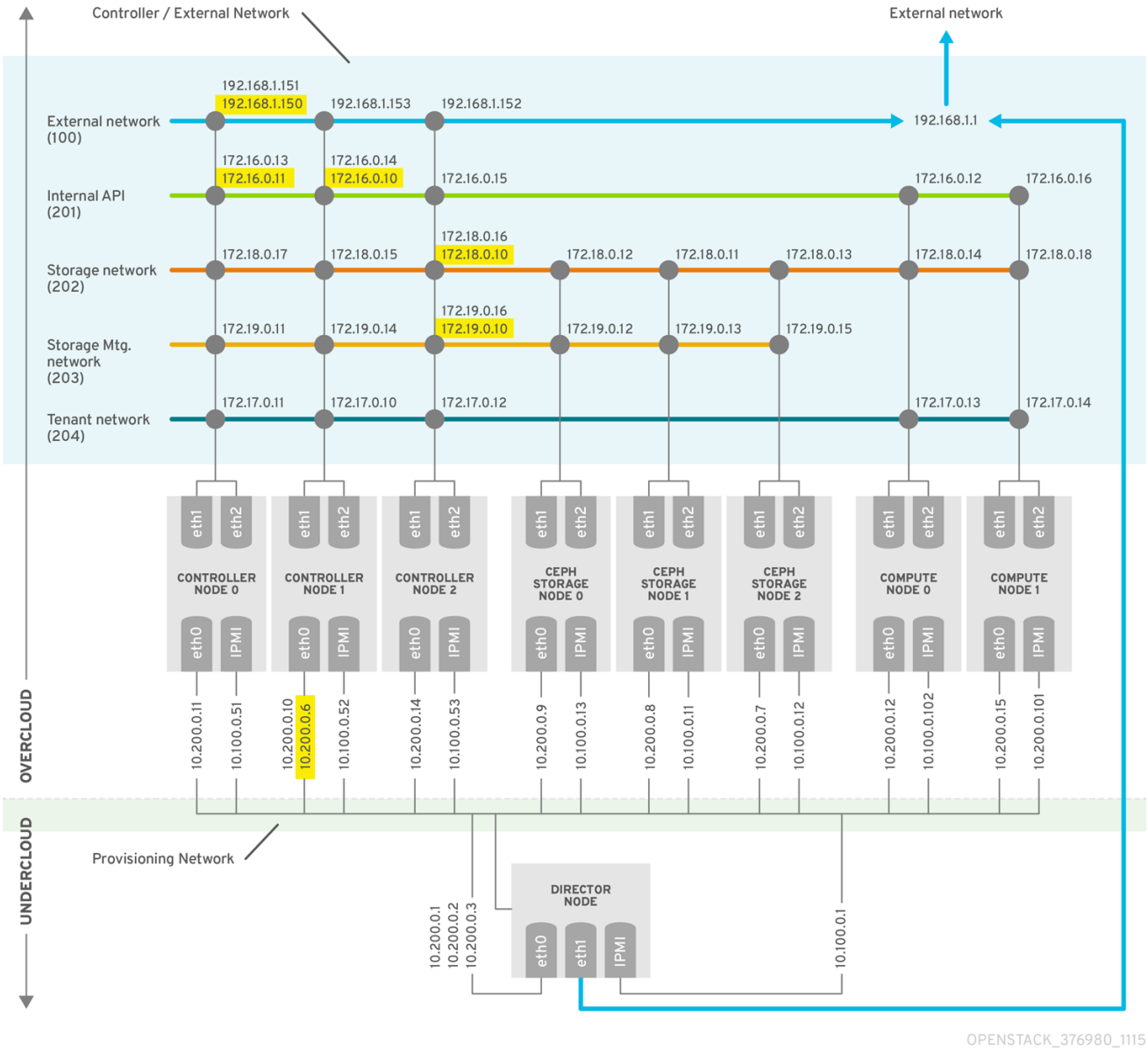
第 2 章 示例部署：带有计算和 CEPH 的高可用性集群

以下示例场景显示了使用 OpenStack 计算服务和 Red Hat Ceph Storage 进行高可用性部署的架构、硬件和网络规格，以及 undercloud 和 overcloud 配置文件。

重要

此部署旨在作为测试环境的引用，在生产环境中不被支持。

图 2.1. 高可用性部署架构示例



有关部署 Red Hat Ceph Storage 集群的更多信息，请参阅使用容器化 Red Hat Ceph 部署 Overcloud 。

有关使用 director 部署 Red Hat OpenStack Platform 的更多信息，请参阅 Director 安装和使用。

2.1. 硬件规格

下表显示了示例部署中使用的硬件。您可以根据需要在自己的测试部署中调整 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	6144 MB	40 GB	IPMI	2（1 个外部；1，在调配时），+ 1 IPMI
3	Controller 节点	4	6144 MB	40 GB	IPMI	3（overcloud 上的 2 绑定；调配时 1 个绑定）+ 1 IPMI
3	Ceph Storage 节点	4	6144 MB	40 GB	IPMI	3（overcloud 上的 2 绑定；调配时 1 个绑定）+ 1 IPMI
2	Compute 节点（根据需要添加更多）	4	6144 MB	40 GB	IPMI	3（overcloud 上的 2 绑定；调配时 1 个绑定）+ 1 IPMI

在规划硬件分配时，请查看以下指南：

Controller 节点

大多数非存储服务在 Controller 节点上运行。所有服务在三个节点之间复制，并且配置为主动或主动 - 被动服务。HA 环境需要至少三个节点。

Red Hat Ceph Storage nodes

存储服务在这些节点上运行，并为计算节点提供 Red Hat Ceph Storage 区域池。至少三个节点。

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，以满足您的容量要求，以及迁移和重新引导操作。您必须将 Compute 节点连接到存储网络和租户网络，以确保虚拟机可以访问存储节点、其他 Compute 节点上的虚拟机以及公共网络。

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。

2.2. 网络规格

下表显示了示例部署中使用的网络配置。

注意

这个示例不包括 control plane 的硬件冗余，以及配置了 overcloud keystone 管理端点的 provisioning 网络。

表 2.2. 物理和虚拟网络

物理 NIC	用途	VLANs	描述
eth0	置备网络(undercloud)	N/A	从 director(undercloud) 管理所有节点
eth1 和 eth2	Controller/External(overcloud)	N/A	使用 VLAN 的绑定 NIC
	外部网络	VLAN 100	允许从环境到租户网络、内部 API 和 OpenStack Horizon 控制面板的访问
	内部 API	VLAN 201	提供对 Compute 节点和 Controller 节点之间的内部 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
	租户网络	VLAN 204	为 RHOSP 提供租户网络服务

除了网络配置外，还必须部署以下组件：

置备网络交换机

- 此切换必须能够将 undercloud 连接到 overcloud 中的所有物理计算机。
- 连接到此切换的每个 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在部署中为其他 VLAN 执行 VLAN 标记。
- 只允许 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。

2.3. UNDERCLOUD 配置文件

示例部署使用了以下 undercloud 配置文件。

instackenv.json

```
{
    "nodes": [
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.11",
        "mac": [
          "2c:c2:60:3b:b3:94"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
      },
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.12",
        "mac": [
          "2c:c2:60:51:b7:fb"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
      },
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.13",
        "mac": [
          "2c:c2:60:76:ce:a5"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
      },
      {
        "pm_password": "testpass",
        "memory": "6144",
```

```
    "pm_addr": "10.100.0.51",
    "mac": [
      "2c:c2:60:08:b1:e2"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.52",
    "mac": [
      "2c:c2:60:20:a1:9e"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.53",
    "mac": [
      "2c:c2:60:58:10:33"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.101",
    "mac": [
      "2c:c2:60:31:a9:55"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "2",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.102",
    "mac": [
      "2c:c2:60:0d:e7:d1"
    ],
```

```
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "2",
        "pm_user": "admin"
       }
    ],
    "overcloud": {"password": "7adbbbeedc5b7a07ba1917e1b3b228334f9a2d4e",
    "endpoint": "http://192.168.1.150:5000/v2.0/"
              }
}
```

undercloud.conf

```
[DEFAULT]
image_path = /home/stack/images
local_ip = 10.200.0.1/24
undercloud_public_vip = 10.200.0.2
undercloud_admin_vip = 10.200.0.3
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
local_interface = eth0
masquerade_network = 10.200.0.0/24
dhcp_start = 10.200.0.5
dhcp_end = 10.200.0.24
network_cidr = 10.200.0.0/24
network_gateway = 10.200.0.1
#discovery_interface = br-ctlplane
discovery_iprange = 10.200.0.150,10.200.0.200
discovery_runbench = 1
undercloud_admin_password = testpass
...
```

network-environment.yaml

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/cinder-
storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/swift-
storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/ceph-
storage.yaml

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ExternalNetCidr: 192.168.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end': '172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end': '172.19.0.200'}]
  # Leave room for floating IPs in the External allocation pool
```

```
  ExternalAllocationPools: [{'start': '192.168.1.150', 'end': '192.168.1.199'}]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ExternalNetworkVlanID: 100
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.1.1
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: """"
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=active-backup lacp=off other_config:bond-miimon-interval=100"
```

2.4. OVERCLOUD 配置文件

示例部署使用了以下 overcloud 配置文件。

/var/lib/config-data/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（Controller 节点）

此文件定义集群基础架构，并可在所有 Controller 节点上使用。

```
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
```

第 3 章 访问高可用性环境

您可以从 undercloud 访问和调查特定的 HA 节点。

流程

1. 在运行 HA 环境中，登录 undercloud。
2. 进入 **stack** 用户：

```
# sudo su - stack
```

3. 要访问和调查 undercloud 节点，请从 undercloud 获取节点的 IP 地址：

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

4. 要登录到其中一个 overcloud 节点，请运行以下命令：

```
(undercloud) $ source ~/overcloudrc
(overcloud) $ ssh [NODE_NAME]@[NODE_IP]
```

将名称和 IP 地址替换为部署的实际值。

第 4 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、RabbitMQ、Redis 和 HAProxy。您可以使用 Pacemaker 查看和管理受管服务、虚拟 IP 地址、电源管理和隔离的一般信息。

有关 Red Hat Enterprise Linux 中的 Pacemaker 的详情，请参考 Red Hat Enterprise Linux 文档中的 配置和管理高可用性集群。

4.1. 资源捆绑包和容器

Pacemaker 将 Red Hat OpenStack Platform(RHOSP)服务作为 *捆绑设置资源或 捆绑包 进行管理*。大多数服务都是以相同方式启动且始终在每个 Controller 节点上运行的主动服务。

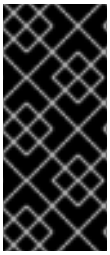
Pacemaker 管理以下资源类型：

捆绑包（Bundle）

捆绑包资源在所有 Controller 节点上配置和复制相同的容器，映射容器目录所需的存储路径，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 HAProxy 等简单 **systemd** 服务到 Galera 等复杂服务，这需要控制和设置不同节点上的服务状态。



重要

- 您不能使用 **docker** 或 **systemctl** 管理捆绑包或容器。您可以使用 命令检查服务的状态，但必须使用 Pacemaker 对这些服务执行操作。
- Pacemaker 控制的 Docker 容器将 **RestartPolicy** 设置为 **no**。这是为了确保 Pacemaker（而非 Docker）控制容器启动和停止操作。

简单捆绑包设置资源（简单捆绑包）

简单的 Bundle Set 资源或 *简单的捆绑包* 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：

```
Docker container set: haproxy-bundle [192.168.24.1:8787/rhosp-rhel7/openstack-
haproxy:pcmklatest]
  haproxy-bundle-docker-0    (ocf::heartbeat:docker):    Started overcloud-controller-0
  haproxy-bundle-docker-1    (ocf::heartbeat:docker):    Started overcloud-controller-1
  haproxy-bundle-docker-2    (ocf::heartbeat:docker):    Started overcloud-controller-2
```

对于每个捆绑包，您可以看到以下详情：

- Pacemaker 分配给该服务的名称。
- 对与捆绑包关联的容器的引用。
- 在不同 Controller 节点上运行的副本的列表和状态。

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
 Docker: image=192.168.24.1:8787/rhosp-rhel7/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)
```

以下示例显示了以下与捆绑包中容器相关的信息：

- **镜像** ：供容器使用的镜像，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** ：指示容器的数量必须在集群中运行。每个捆绑包包括三个容器，每个 Controller 节点对应一个容器。
- **run-command** ：用于生成容器的 System 命令。
- **存储映射** ：将每个主机上的本地路径映射到容器。要从主机检查 **haproxy** 配置，打开 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 文件，而不是 **/etc/haproxy/haproxy.cfg** 文件。

注意

虽然 HAProxy 通过负载均衡流量到所选服务来提供高可用性服务，但您可以将 HAProxy 配置为高度可用的服务，方法是将它作为 Pacemaker 捆绑服务进行管理。

复杂的 Bundle Set resources（complex 捆绑包）

复杂的 Bundle Set 资源或 *复杂的捆绑包* 是 Pacemaker 服务，除了包含在简单捆绑包中的基本容器配置外，还指定资源配置。

需要此配置来管理 *多State* 资源，它们是服务，根据它们运行的 Controller 节点，它们有不同的状态。

这个示例显示 **pcs status** 命令的输出中的复杂捆绑包列表：

```
Docker container set: rabbitmq-bundle [192.168.24.1:8787/rhosp-rhel7/openstack-
rabbitmq:pcmklatest]
  rabbitmq-bundle-0    (ocf::heartbeat:rabbitmq-cluster):     Started overcloud-controller-0
  rabbitmq-bundle-1    (ocf::heartbeat:rabbitmq-cluster):     Started overcloud-controller-1
  rabbitmq-bundle-2    (ocf::heartbeat:rabbitmq-cluster):     Started overcloud-controller-2
```

```
Docker container set: galera-bundle [192.168.24.1:8787/rhosp-rhel7/openstack-mariadb:pcmklatest]
  galera-bundle-0      (ocf::heartbeat:galera):        Master overcloud-controller-0
  galera-bundle-1      (ocf::heartbeat:galera):        Master overcloud-controller-1
  galera-bundle-2      (ocf::heartbeat:galera):        Master overcloud-controller-2
Docker container set: redis-bundle [192.168.24.1:8787/rhosp-rhel7/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
```

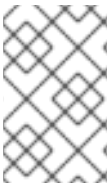
此输出显示以下有关每个复杂捆绑包的信息：

- RabbitMQ：所有三个 Controller 节点都运行服务的独立实例，类似于简单的捆绑包。
- Galera：所有三个 Controller 节点在同一限制下作为 Galera 主节点运行。
- 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。每个容器类型可能会在不同的约束下运行。

以下示例显示了 **galera-bundle** 复杂捆绑包的设置：

```
[...]
Bundle: galera-bundle
 Docker: image=192.168.24.1:8787/rhosp-rhel7/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，与简单的捆绑包不同，**galera-bundle** 资源包含了明确的资源配置，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 4.4 节 “查看虚拟 IP 地址”。

4.2. 查看常规 PACEMAKER 信息

要查看常规 Pacemaker 信息，请使用 **pcs status** 命令。

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 1.1.16-12.el7_4.5-94ff4df) - partition with quorum

Last updated: Thu Feb  8 14:29:21 2018
Last change: Sat Feb  3 11:37:17 2018 by root via cibadmin on overcloud-controller-2

12 nodes configured
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以下有关集群的信息：

- **集群名称** ：集群名称。
- **[NUM] 节点配置** ：为集群配置的节点数。
- **[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** ：当前在线的客户端节点的名称。每个客户机节点由一个复杂的 Bundle Set 资源组成。有关捆绑包集的详情，请参考 第 4.1 节 “资源捆绑包和容器”。

4.3. 查看捆绑包状态

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。

从 undercloud 节点检查捆绑包状态

运行以下命令：

```
$ sudo docker exec -it haproxy-bundle-docker-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。

从 Controller 节点检查捆绑包状态

登录 Controller 节点并运行以下命令：

```
$ 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
```

4.4. 查看虚拟 IP 地址

每个 IPaddr2 资源设置虚拟 IP 地址，客户端使用它来请求访问服务。如果 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。

显示所有虚拟 IP 地址

使用 **--full** 选项运行 **pcs resource show** 命令来显示使用 **VirtualIP** 类型的所有资源：

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

以下示例输出显示当前设置为侦听特定虚拟 IP 地址的每个 Controller 节点：

```
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 地址最初附加到特定的 Controller 节点。例如，**overcloud-controller-0** 上启动 **192.168.1.150**。但是，如果该 Controller 节点失败，IP 地址会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，并显示了每个 IP 地址的原始分配。

表 4.1. IP 地址描述和分配源

IP 地址	描述	分配自
192.168.1.150	公共 IP 地址	**network-environment.yaml** 文件中的 **ExternalAllocationPools** 属性
10.200.0.6	控制器虚拟 IP 地址	**undercloud.conf** 文件中的 **dhcp_start** 和 **dhcp_end** 范围部分设置为 **10.200.0.5-10.200.0.24**
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**
172.18.0.10	存储虚拟 IP 地址，提供对 Glance API 和 Swift 代理服务的访问	**network-environment.yaml** 文件中的 **StorageAllocationPools** 属性
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**
172.19.0.10	提供对存储管理的访问	**network-environment.yaml** 文件中的 **StorageMgmtAlloctionPools**

查看特定 IP 地址

运行 **pcs resource show** 命令。

```
$ 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)
```

查看特定 IP 地址的网络信息

1. 登录到分配给您要查看的 IP 地址的 Controller 节点。

2. 运行 **ip addr show** 命令来查看网络接口信息。

```
$ ip addr show vlan100
```

输出示例：

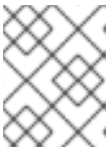
```
 9: vlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN
   link/ether be:ab:aa:37:34:e7 brd ff:ff:ff:ff:ff:ff
   inet *192.168.1.151/24* brd 192.168.1.255 scope global vlan100
      valid_lft forever preferred_lft forever
   inet *192.168.1.150/32* brd 192.168.1.255 scope global vlan100
      valid_lft forever preferred_lft forever
```

3. 运行 **netstat** 命令，以显示侦听 IP 地址的所有进程。

```
$ 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** 获取。这些进程包括 **sshd**、**mysqld**、**dhclient**、**ntpd**。

查看端口号分配

打开 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 文件，以查看默认端口号分配。

以下示例显示了端口号及其侦听的服务：

- TCP 端口 6080： **nova_ ever**
- 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，在 **haproxy.cfg** 文件中定义的大多数服务都侦听所有三个 Controller 节点上的 **192.168.1.150** IP 地址。但是，只有 **controller-0** 节点在外部侦听 **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，所有其他服务则已在回退 Controller 节点上运行。

4.5. 查看 PACEMAKER 状态和电源管理信息

pcs status 输出的最后一个部分显示您的电源管理隔离（如 IPMI）的信息，以及 Pacemaker 服务本身的状态：

```
 my-ipmilan-for-controller-0 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-0
 my-ipmilan-for-controller-1 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-1
 my-ipmilan-for-controller-2 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-2

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

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled openstack-cinder-volume        (systemd:openstack-cinder-volume):
Started overcloud-controller-0

  pcsd: active/enabled
```

my-ipmilan-for-controller 设置显示每个 Controller 节点(**stonith:fence_ipmilan**)的隔离类型，以及 IPMI 服务是否已停止或正在运行。PCSD 状态显示所有三个 Controller 节点目前在线。Pacemaker 服务包含三个守护进程： **corosync**、**pacemaker** 和 **pcsd**。在示例中，所有三个服务都处于活跃状态并已启用。

4.6. 失败的 PACEMAKER 资源故障排除

如果一个 Pacemaker 资源失败，您可以查看 **pcs status** 输出的 **Failed Actions** 部分。在以下示例中，**openstack-cinder-volume** 服务停止处理 **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
```

在这种情况下，您必须启用 systemd 服务 *openstack-cinder-volume*。在其他情况下，您可能需要查找并修复问题，然后清理资源。有关排除资源问题的更多信息，请参阅 第 8 章 *资源问题的故障排除*。

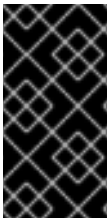
第 5 章 使用 STONITH 的隔离 CONTROLLER 节点

隔离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，故障的节点可能会导致集群中的数据崩溃。

director 使用 Pacemaker 提供高度可用的 Controller 节点集群。Pacemaker 使用名为 *STONITH* 的进程来隔离出现故障的节点。STONITH 是 "Shoot the node in the head" 的缩写。

如果 Controller 节点失败，作为 Pacemaker 指定的协调器(DC)的 Controller 节点会使用 Pacemaker **stonith** 服务来隔离受影响的 Controller 节点。

默认情况下禁用 STONITH，需要手动配置，以便 Pacemaker 可以控制集群中每个节点的电源管理。



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。

有关在 Red Hat Enterprise Linux 中使用 Pacemaker 进行隔离的更多信息，请参阅：

- 高可用性附加组件概述
- High Availability Add-On 管理
- High Availability Add-On 参考

5.1. 支持的隔离代理

当您使用隔离部署高可用性环境时，您可以根据您的环境需要选择以下隔离代理之一。要更改隔离代理，您必须在 fence **.yaml** 文件中配置附加参数，如 第 5.2 节 “在 overcloud 上部署和测试隔离” 所述。

智能平台管理接口 (IPMI)

RHOSP 用于管理隔离的默认隔离机制。

存储块设备(SBD)

在部署中使用 Watchdog 设备。部署不得使用共享存储。

fence_kdump

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

除了 IPMI、**fence_rhevm** 或 Redfish 隔离代理外，您还可以将这个代理配置为辅助机制。如果您配置多个隔离代理，请确保分配足够时间以便第一个代理完成该任务，然后再启动下一个任务。

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** 参数的值更改为 **fence_redfish**。有关 Redfish 的更多信息，请参阅 DTMF 文档。

fence_rhevm for Red Hat Virtualization (RHV)

使用 为运行 RHV 环境的 Controller 节点配置隔离。您可以以与 IPMI **隔离相同的方式生成 fence.yaml** 文件，但您必须在 **nodes.json** 文件中定义 **pm_type** 参数才能使用 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

确保您使用具有权限的角色在 RHV 中创建和启动虚拟机，如 **UserVMManager**。

5.2. 在 OVERCLOUD 上部署和测试隔离

隔离配置过程包括以下阶段：

1. 查看 STONITH 和 Pacemaker 的状态。
2. 生成 **fencing.yaml** 文件。
3. 重新部署 overcloud 并测试配置。

先决条件

确保您可以访问您在 director 中注册 Controller 节点时创建的 **nodes.json** 文件。此文件是您在部署过程中生成的 **fence.yaml** 文件所需的输入。

检查 STONITH 和 Pacemaker 的状态

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：

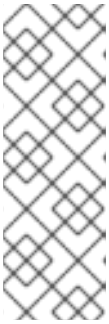
```
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

生成 fence.yaml 环境文件

选择以下选项之一：

- 如果使用 IPMI 或 Red Hat Virtualization(RHV)隔离代理，请运行以下命令来生成 **fence.yaml** 环境文件：

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



注意

- 此命令将 **ilo** 和 **drac** 电源管理详细信息转换为 IPMI 等效值。
- 确保 **nodes.json** 文件包含节点上其中一个网络接口的 MAC 地址。有关更多信息，请参阅 Overcloud 的注册节点。
- 如果使用 RHV，请确保使用具有权限的角色来创建和启动虚拟机，如 **UserVMManager**。

- 如果您使用不同的隔离代理，如 Storage Block Device(SBD)、**fence_kdump** 或 Redfish，请手动生成 **fence.yaml** 文件。

注意

如果使用预置备节点，还必须手动创建 **fence.yaml** 文件。

有关支持的隔离代理的详情，请参考 第 5.1 节 “支持的隔离代理”。

重新部署 overcloud 并测试配置

1. 运行 **overcloud 部署命令** 并包含您生成的 **fence.yaml** 文件，以便在 Controller 节点上配置隔离：

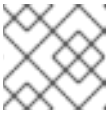
```
openstack overcloud deploy --templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e ~/templates/network-environment.yaml \
-e ~/templates/storage-environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-scale 3 --control-flavor control --compute-flavor Compute --ceph-storage-flavor ceph-storage --ntp-server pool.ntp.org --neutron-network-type vxlan --neutron-tunnel-types vxlan \
-e fencing.yaml
```

2. 登录到 overcloud，并验证是否为每个 Controller 节点配置了隔离：

   a. 检查 Pacemaker 是否已配置为资源管理器：

```
$ source stackrc
$ nova 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 被配置为为 **fence.yaml** 文件中指定的每个 Controller 节点使用 STONITH 资源。



注意

您不能在其控制的同一节点中配置 **fence-resource** 进程。

b. 运行 **pcs stonith show** 命令检查隔离资源属性：

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

STONITH 属性值必须与 **fence.yaml 文件中的值匹配**。

验证 Controller 节点上的隔离

要测试隔离是否正常工作，您可以通过关闭 Controller 节点上的所有端口并重启服务器来触发隔离。

1. 登录到 Controller 节点：

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

2. 切换到 **root** 用户并在每个端口中运行 **iptables** 命令：

```
$ 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
```

重要

此步骤丢弃到 Controller 节点的所有连接，这会导致服务器重启。

3. 在不同的 Controller 节点中找到 Pacemaker 日志文件的隔离事件：

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

如果 STONITH 服务对控制器执行隔离操作，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** 命令验证重新引导的 Controller 节点是否已在集群中再次运行。

5.3. 查看 STONITH 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：

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

```
 Resource: my-ipmilan-for-controller-0 (class=stonith type=fence_ipmilan) (1)
  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)
```

--full 选项返回三个 Controller 节点的隔离详情。

此输出显示每个资源的以下信息：

- 隔离设备用来打开和关闭机器的 IPMI 电源管理服务，如 **fence_ipmilan**。
- IPMI 接口的 IP 地址，如 **10.100.0.51**。
- 使用 登录的用户名，如 **admin**。
- 用于登录到节点的密码，如 **abc**。
- 监控每个主机的间隔（如 **60s** ）。

5.4. 隔离参数

当您在 overcloud 上部署隔离时，会生成一个带有所需参数的 **fence.yaml** 文件来配置隔离。有关部署和测试隔离的详情，请参考 第 5.2 节 “在 overcloud 上部署和测试隔离”。

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。Red Hat OpenStack Platform 仅支持 IPMI 的 **fence_ipmilan**。
- **host_mac** ：位于 provisioning 接口的小写或服务器上任何其他网络接口的 mac 地址。您可以将此用作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

- **auth**: IPMI 身份验证类型（**md5**、**password** 或 none）。
- **ipaddr** ：IPMI IP 地址.
- **ipport** ：IPMI 端口。
- **Login**: IPMI 设备的 Username。
- **passwd** ：IPMI 设备的密码。
- **特兰加** ：使用 lanplus 提高连接安全性。
- **privlvl**: IPMI 设备上的特权级别
- **pcmk_host_list** ：Pacemaker 主机列表。

第 6 章 使用 HAPROXY 负载均衡流量

HAProxy 服务为高可用性集群中的 Controller 节点提供流量负载平衡，以及记录和示例配置。

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 服务作为名为 **haproxy-bundle** 的高可用性服务进行管理。

有关 HAProxy 的更多信息，请参阅 Load Balancer 管理指南中的 HAProxy 配置 章节。

有关验证 HAProxy 是否已正确配置的详情，请参阅 KCS 文章 How can verify my haproxy.cfg 是否已正确配置为负载平衡 openstack 服务？

6.1. 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/haproxy/etc/haproxy/haproxy.cfg** 文件中配置这些服务，它会指示 HAProxy 在每个 overcloud 节点上运行。

下表显示了 HAProxy 管理的服务列表：

表 6.1. 由 HAProxy 管理的服务

Nhce	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：

- **侦听** ：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**:内部 API 网络(VLAN201)上的虚拟 IP 地址和端口在 overcloud 中使用。
- **192.168.1.150:8776**: External Network(VLAN100)上的虚拟 IP 地址和端口提供从 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** ：控制器节点名称和 IP 地址。HAProxy 可将请求定向到这些 IP 地址到 **服务器** 输出中列出的其中一个 Controller 节点。
- **httpchk** ：在 Controller 节点服务器上启用健康检查。
- **的第 5 分**：失败健康检查数以确定服务离线。
- **2000 年间** ：连续两次健康检查之间的间隔，以毫秒为单位。
- **出现了** 2：成功健康检查的数量，以确定该服务正在运行。

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。

6.2. 查看 HAPROXY 统计数据

默认情况下，director 还在所有 HA 部署中启用 HAProxy Stats 或统计数据。使用此功能，您可以查看 HAProxy Stats 页面中的数据传输、连接和服务器状态的详细信息。

director 还设置用于访问 HAProxy Stats 页面的 **IP:Port** 地址，并将信息存储在 **haproxy.cfg** 文件中。

流程

1. 在安装 HAProxy 的任何 Controller 节点中打开 **/var/lib/config-data/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. 在网页浏览器中，导航到 **10.200.0.6:1993**，并从 **stats auth** 行中输入凭证来查看 HAProxy Stats 页面。

第 7 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 服务作为管理数据库 master/slave 状态 **的捆绑包设置** 资源运行。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。

和其它 Pacemaker 服务类似，您可以使用 **pcs status** 命令检查 Galera 服务正在运行，以及它正在运行的 Controller 节点。有关查看 Pacemaker 捆绑包状态的更多信息，请参阅 第 4.3 节 “查看捆绑包状态”。

在调查数据库集群完整性时，每个节点都必须满足以下条件：

- 节点是正确集群的一部分。
- 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。
- 节点是将 write-sets 复制到本地数据库中表。

7.1. 验证主机名解析

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

要对 MariaDB Galera 集群进行故障排除，您可以首先消除任何主机名解析问题，然后在每个 Controller 节点的数据库上检查 write-set 复制状态。要访问 MySQL 数据库，您要在 overcloud 部署期间使用 director 设置的密码。

流程

1. 在 Controller 节点上，通过运行 **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 docker ps | grep -i galera
5fb195b0d9e8        192.168.24.1:8787/rh-osbs/rhosp13-openstack-mariadb:pcmklatest
"dumb-init -- /bin..."   7 hours ago         Up 7 hours  galera-bundle-docker-0
```

3. 从每个节点上的 MariaDB 数据库获取写入设置复制信息。

```
$ sudo docker exec galera-bundle-docker-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. 通过检查集群报告正确的节点数量来验证 MariaDB Galera 集群的健康状态和完整性。

7.2. 检查数据库集群完整性

当您调查 MariaDB Galera 集群的问题时，您可以通过检查每个 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 ***VARIABLE*** 替换为您要检查的 **wsrep** 数据库变量：

```
$ sudo docker exec galera-bundle-docker-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 docker exec galera-bundle-docker-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 7.1. 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	集群中的节点数量	您可以在任何节点上检查它。如果该值小于实际节点数，那么某些节点将出现故障或丢失连接。

变量	概述	描述
wsrep_cluster_conf_id	集群更改总数	确定集群是否被分成几个组件，也称为 *分区*。分区通常由网络故障造成的。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（**主要**）。
wsrep_cluster_status	主要组件状态	决定节点是否可以写入集群。如果节点可以写入集群，**wsrep_cluster_status** 值是 **Primary**。任何其他值都表示节点是非操作分区的一部分。

7.3. 检查数据库节点完整性

如果您可以将 Galera 集群问题隔离到特定的节点，某些 **wsrep** 数据库变量可能会指示节点中的特定问题。

流程

运行以下命令并将 ***VARIABLE*** 替换为您要检查的 **wsrep** 数据库变量：

```
$ sudo docker exec galera-bundle-docker-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 7.2. 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集群中的 write-sets。如果是这样，那么 **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中的其他节点。如果是这样，那么 **wsrep_connected** 为 **ON**。

变量	概述	描述
wsrep_local_state_comment	节点状态	总结节点状态。如果节点可写入集群，那么 **wsrep_local_state_comment** 的典型值可以是 **Joining**,**Waiting on SST**、**Joined**、**Synced** 或 **Donor**。 如果节点是非操作组件的一部分，那么 **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，也会有 **wsrep_connected** 值。**例如，如果集群分区，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 7.2 节 “检查数据库集群完整性”。**
- **如果 wsrep_connected 值为 OFF，则该节点没有连接到任何集群组件。**

7.4. 测试数据库复制性能

如果集群和单个节点都健康且稳定，您可以通过查询特定的数据库变量，在复制吞吐量上运行性能基准测试。

每次查询其中一个变量时，FLUSH STATUS 命令都会重置变量值。要运行基准测试，您必须运行多个查询和分析差异。这些差异可帮助您确定正在运行的 *流控制* 是否影响到集群性能。

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，流控制会暂停复制，直到队列大小停机。有关 Flow Control 的更多信息，请参阅 Galera Cluster 网站上的流控制。

流程

运行以下命令并将 ***VARIABLE*** **替换为您要检查的 wsrep 数据库变量：**

```
$ sudo docker exec galera-bundle-docker-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 7.3. 用于检查数据库复制性能的数据库变量

变量	概述	使用
wsrep_local_recv_queue_avg	最后一次查询后本地接收的 write-set 队列的平均大小。	高于 0.0 的值表示节点无法像接收 write-sets 时快速应用 write-sets，这会触发复制节流。检查 **wsrep_local_recv_queue_min** 和 **wsrep_local_recv_queue_max** 查看这个基准的详细查看。
wsrep_local_send_queue_avg	最后一次查询后的平均发送队列长度。	值高于 0.0 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** 的值大于 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部分。	高于 0.0 的值表示流控制暂停该节点。要确定暂停的持续时间，请在查询之间使用以秒为单位的 **wsrep_flow_control_paused** 值。最佳值是尽可能接近 0.0。 例如： ● 如果在最后一次查询后 **wsrep_flow_control_paused** 的值为 0.50 1 分钟，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 Flow Control 会暂停整个一分钟的节点。
wsrep_cert_deps_distance	可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，这个变量指示平均可并行应用多少个 write-sets。将该值与 **wsrep_slave_threads** 变量进行比较，以查看实际可同时应用了多少 write-sets。

变量	概述	使用
wsrep_slave_threads	可同时应用的线程数	您可以增加这个变量的值来同时应用更多线程，这也会增加 **wsrep_cert_deps_distance** 的值。**wsrep_slave_threads** 的值不得高于节点中 CPU 内核数。 例如，如果 **wsrep_cert_deps_distance** 值是 **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入设置量。 如果有问题的节点已经有最佳 **wsrep_slave_threads** 值，您可以在调查可能的连接问题时从集群中排除该节点。

第 8 章 资源问题的故障排除

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。根据您的部署，资源失败的原因有很多，您必须调查资源来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，并且资源可以互相连接。您还可以检查比其他 Controller 节点更频繁隔离的 Controller 节点，以识别可能的通信问题。

8.1. 查看资源限制

您可以查看如何启动服务的限制，包括与每个资源所在的位置相关的限制、资源启动顺序以及资源是否必须与另一个资源共存。

查看所有资源限制

在任何 Controller 节点上，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 命令中截断的输出：

```
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)
```

这个输出显示以下主要的约束类型：

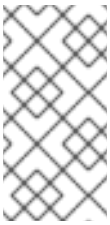
位置约束

列出可分配资源的位置：

- 第一个约束定义了一个规则，它将 galera-bundle 资源设置为在 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，这是使服务可访问的必要。
- 第三个位置约束显示 ipmilan 资源在每个 Controller 节点上被禁用。

排序限制

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



注意

排序限制仅适用于 IP 地址资源和 HAproxy。systemd 管理所有其他资源，因为 Compute 等服务应该出现相依服务的中断，如 Galera。

colocation Constraints

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

查看 Galera 位置约束

在任何 Controller 节点上，运行 pcs attribute show 命令。

```
$ sudo pcs property show
```

输出示例：

```
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: tripleo_cluster
 dc-version: 1.1.16-12.el7_4.5-94ff4df
 have-watchdog: false
 redis_REPL_INFO: overcloud-controller-0
 stonith-enabled: false
Node Attributes:
 overcloud-controller-0: cinder-volume-role=true galera-role=true haproxy-role=true rabbitmq-
role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-0
 overcloud-controller-1: cinder-volume-role=true galera-role=true haproxy-role=true rabbitmq-
role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-1
 overcloud-controller-2: cinder-volume-role=true galera-role=true haproxy-role=true rabbitmq-
role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-2
```

在此输出中，您可以验证所有 Controller 节点的 galera-role 属性是否为 true。这意味着 galera-bundle 资源仅在这些节点上运行。同样的概念适用于与其他位置限制关联的其他属性。

8.2. 检查 CONTROLLER 节点资源问题

根据问题的类型和位置，您可以采用不同的方法来调查并修复资源。

检查 Controller 节点问题

如果对 Controller 节点的健康检查失败，这可能代表 Controller 节点之间的通信问题。要调查，请登录 Controller 节点并检查服务是否可以正确启动。

调查单个资源问题

如果控制器上的大部分服务都正确运行，您可以运行 pcs status 命令并检查输出以了解有关特定服务故障的信息。您还可以登录到资源失败的 Controller，并调查 Controller 节点上的资源行为。

流程

以下流程演示了如何调查 openstack-cinder-volume 资源。

1. **找到并登录到资源出现故障的 Controller 节点。**

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

```
[heat-admin@overcloud-controller-0 ~]$ sudo systemctl status openstack-cinder-volume
● openstack-cinder-volume.service - Cluster Controlled openstack-cinder-volume
   Loaded: loaded (/usr/lib/systemd/system/openstack-cinder-volume.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/system/openstack-cinder-volume.service.d
           └─50-pacemaker.conf
   Active: active (running) since Tue 2016-11-22 09:25:53 UTC; 2 weeks 6 days ago
 Main PID: 383912 (cinder-volume)
   CGroup: /system.slice/openstack-cinder-volume.service
           ├─383912 /usr/bin/python2 /usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-
dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/volume.log
           └─383985 /usr/bin/python2 /usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-
dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/volume.log


Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.798 383912 WARNING oslo_config.cfg [req-8f32db96-7ca2-4fc5-82ab-
271993b28174 - - - -...e future.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.799 383912 WARNING oslo_config.cfg [req-8f32db96-7ca2-4fc5-82ab-
271993b28174 - - - -...e future.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.926 383985 INFO cinder.coordination [-] Coordination backend started
successfully.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.926 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...r (1.2.0)
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.047 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.048 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.048 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.063 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...essfully.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.111 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...r (1.2.0)
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.146 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...essfully.
Hint: Some lines were ellipsized, use -l to show in full.
```

3. **根据输出中的信息更正失败的资源。**

4. 运行 **pcs resource cleanup** 命令以重置状态以及资源失败计数。

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

第 9 章 监控高可用性 RED HAT CEPH STORAGE 集群

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。

查看 Ceph Monitoring 服务的状态

在 Controller 节点上，运行 service ceph status 命令来检查 Ceph Monitoring 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"}
```

查看 Ceph 监控配置

在 Controller 节点或 Ceph 节点上，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
```

本例显示以下信息：

- 所有三个 Controller 节点都被配置为使用 mon_initial_members 参数监控 Red Hat Ceph Storage 集群。
- 172.19.0.11/24 网络配置为提供 Controller 节点和 Red Hat Ceph Storage 节点之间的通信路径。
- Red Hat Ceph Storage 节点分配给与 Controller 节点独立的网络，而监控 Controller 节点的 IP 地址为 172.18.0.15、172.18.0.16 和 172.18.0.17。

查看单独的 Ceph 节点状态

登录 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 参数值是 HEALTH_OK，这表示 Ceph 节点处于活动状态且健康。输出中显示了三个 overcloud-controller 节点上运行的 Ceph 监控服务，以及服务的 IP 地址和端口。

有关 Red Hat Ceph Storage 的更多信息，请参阅 Red Hat Ceph 产品页面。