



Red Hat OpenStack Platform 17.1

管理高可用性服务

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

Red Hat OpenStack Platform 17.1 管理高可用性服务

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

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

为了保持 OpenStack 环境启动并高效运行，请使用 Red Hat OpenStack Platform director 创建配置，以便在 OpenStack 中所有主要服务间提供高可用性和负载均衡。

目录

使开源包含更多	4
对红帽文档提供反馈	5
第 1 章 RED HAT OPENSTACK PLATFORM 高可用性概述和计划	6
1.1. RED HAT OPENSTACK PLATFORM 高可用性服务	6
1.2. 规划高可用性硬件分配	7
1.3. 规划高可用性网络	7
1.4. 访问高可用性环境	7
1.5. 其他资源	8
第 2 章 示例部署：使用 COMPUTE 和 CEPH 的高可用性集群	9
2.1. 高可用性硬件规格示例	9
2.2. 高可用性网络规格示例	10
2.3. 高可用性 UNDERCLOUD 配置文件示例	11
2.4. 高可用性 OVERCLOUD 配置文件示例	14
2.5. 其他资源	20
第 3 章 使用 PACEMAKER 管理高可用性服务	21
3.1. PACEMAKER 资源捆绑包和容器	21
3.2. 检查 PACEMAKER 集群状态	24
3.3. 在高可用性集群中检查捆绑包状态	25
3.4. 在高可用性集群中查看虚拟 IP 的资源信息	25
3.5. 在高可用性集群中查看虚拟 IP 的网络信息	27
3.6. 检查隔离代理和 PACEMAKER 守护进程状态	28
3.7. 其他资源	29
第 4 章 使用 STONITH 隔离 CONTROLLER 节点	30
4.1. 支持的隔离代理	30
4.2. 在 OVERCLOUD 上部署隔离	31
4.3. 测试 OVERCLOUD 上的隔离	34
4.4. 查看 STONITH 设备信息	34
4.5. 隔离参数	35
4.6. 其他资源	36
第 5 章 使用 HAPROXY 负载均衡流量	37
5.1. HAPROXY 的工作原理	37
5.2. 查看 HAPROXY STATS	38
5.3. 其他资源	38
第 6 章 将 OVERCLOUD 配置为使用外部负载均衡器	39
6.1. 为外部负载均衡器准备您的环境	39
6.2. 为外部负载均衡器配置 OVERCLOUD 网络	41
6.3. 创建外部负载均衡器环境文件	42
6.4. 为外部负载均衡配置 SSL	43
6.5. 使用外部负载均衡器部署 OVERCLOUD	44
6.6. 其他资源	46
第 7 章 示例配置：带有外部 HAPROXY 负载均衡器的 OVERCLOUD	47
7.1. HAPROXY 配置文件示例	47
7.2. 使用负载均衡的服务的配置参数	53
第 8 章 使用 GALERA 管理数据库复制	64
8.1. 在 MARIADB 集群中验证主机名解析	64

8.2. 检查 MARIADB 集群完整性	65
8.3. 检查 MARIADB 集群中的数据库节点完整性	66
8.4. 在 MARIADB 集群中测试数据库复制性能	67
8.5. 其他资源	68
第 9 章 对高可用性资源进行故障排除	69
9.1. 在高可用性集群中查看资源限制	69
9.2. 检查 PACEMAKER 资源问题	71
9.3. 调查 SYSTEMD 资源问题	72
第 10 章 监控高可用性 RED HAT CEPH STORAGE 集群	74
10.1. 检查 RED HAT CEPH 监控服务状态	74
10.2. 检查 RED HAT CEPH 监控配置	74
10.3. 检查 RED HAT CEPH 节点状态	74
10.4. 其他资源	75

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](#) 表单在 OpenShift (RHOSO)或更早版本的 Red Hat OpenStack Platform (RHOSP)上提供有关 Red Hat OpenStack Services 文档的反馈。当您为 RHOSO 或 RHOSP 文档创建问题时，这个问题将在 RHOSO Jira 项目中记录，您可以在其中跟踪您的反馈的进度。

要完成 [Create Issue](#) 表单，请确保您已登录到 JIRA。如果您没有红帽 JIRA 帐户，您可以在 <https://issues.redhat.com> 创建一个帐户。

1. 点击以下链接打开 **Create Issue** 页面：[Create Issue](#)
2. 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。
3. 点 **Create**。

第 1 章 RED HAT OPENSTACK PLATFORM 高可用性概述和计划

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。

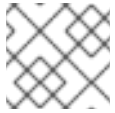
1.1. RED HAT OPENSTACK PLATFORM 高可用性服务

Red Hat OpenStack Platform (RHOSP)使用多种技术来提供实施高可用性(HA)所需的服务。这些服务包括 Galera、RabbitMQ、Red Hat HAProxy、Pacemaker 管理的单个服务，以及 Podman 管理的普通容器服务。

1.1.1. 服务类型

核心容器

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，并需要启动、停止和重启操作的特定管理和限制。您可以使用 Pacemaker 启动、管理核心容器服务并进行故障排除。



注意

RHOSP 使用 [MariaDB Galera 集群](#) 来管理数据库复制。

Active-passive

主动 - 被动服务一次在一个 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预防服务中断的独立服务。因此，如果您重启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。

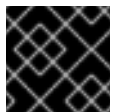
在编排 HA 部署时，director 使用模板和 Puppet 模块来确保正确配置和启动所有服务。另外，当对 HA 问题进行故障排除时，您必须使用 **podman** 命令或 **systemctl** 命令与 HA 框架中的服务交互。

1.1.2. 服务模式

HA 服务可以采用以下模式之一运行：

Active-active

Pacemaker 在多个 Controller 节点上运行相同的 service，并使用 HAProxy 在节点间分发流量，或使用单个 IP 地址将流量分发到特定的 Controller。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。



重要

只有在边缘站点的分布式计算节点(DCN)架构中才支持主动-主动模式。

Active-passive

无法在主动-主动模式运行的服务必须以主动-被动模式运行。在这种模式中，一次只有一个服务实例处于活动状态。例如，HAProxy 使用粘滞选项将传入的 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 Galera 节点同时连接到同一数据。

1.2. 规划高可用性硬件分配

在计划硬件分配时，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)实例的数量。

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，以及迁移和重新引导操作。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](#)。

1.3. 规划高可用性网络

当您规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 中的所有物理计算机。
- 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

- 必须配置此交换机，以便为部署中的其他 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](#)。

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

- 部署并运行高可用性。

流程

1. 在正在运行的 HA 环境中，以 **stack** 用户身份登录 undercloud。
2. 识别 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 tripleo-admin@<node_IP>
```

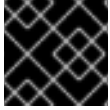
将 **<node_ip>** 替换为您要登录的节点的 IP 地址。

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

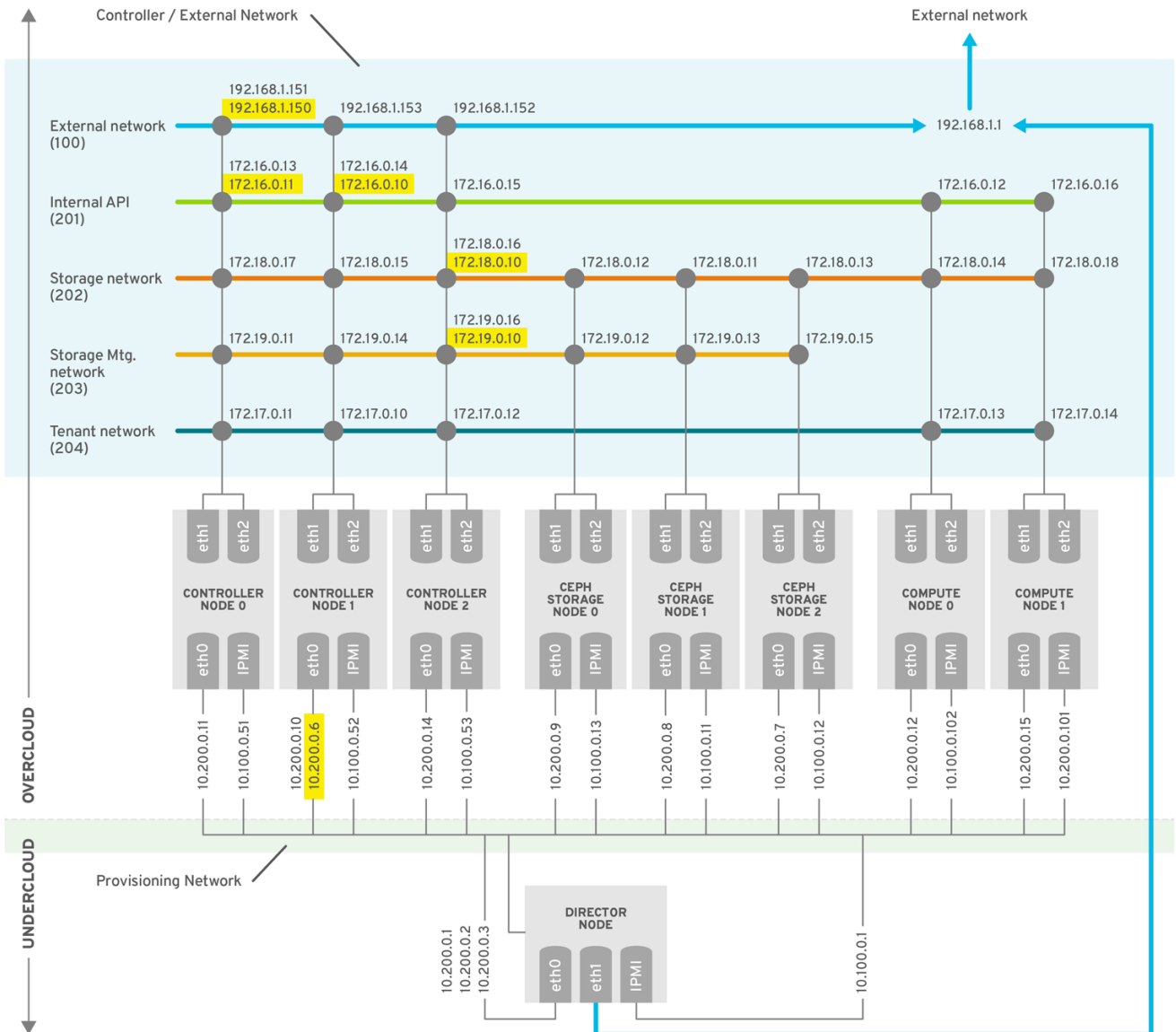
此示例场景演示了架构、硬件和网络规格，以及 undercloud 和 overcloud 配置文件，以用于使用 OpenStack Compute 服务和 Red Hat Ceph Storage 进行高可用性部署。



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



OPENSTACK_376980_1115

2.1. 高可用性硬件规格示例

示例 HA 部署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. 物理机器

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 个外部；调配时 1 个)+ 1IPMI
3	Controller 节点	4	24 GB	40 GB	IPMI	3 (overcloud 上的 2 绑定；调配时 1 个)+ 1IPMI
3	Ceph Storage 节点	4	24 GB	40 GB	IPMI	3 (overcloud 上的 2 绑定；调配时 1 个)+ 1IPMI
2	Compute 节点 (根据需要添加更多)	4	24 GB	40 GB	IPMI	3 (overcloud 上的 2 绑定；调配时 1 个)+ 1IPMI

2.2. 高可用性网络规格示例

HA 部署示例使用特定的虚拟和物理网络配置。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 admin 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	v lans	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
	外部网络	VLAN 100	允许从环境外部访问到项目网络、内部 API 和 OpenStack Horizon 控制面板

物理 NIC	用途	vlan	描述
	内部 API	VLAN 201	提供对 Compute 节点和 Controller 节点之间内部 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 undercloud 配置文件(stackenv.json、undercloud.conf 和 network-environment.yaml)。

instackenv.json

```
{
  "nodes": [
    {
      "pm_password": "testpass",
      "memory": "24",
      "pm_addr": "10.100.0.11",
      "mac": [
        "2c:c2:60:3b:b3:94"
      ],
      "pm_type": "ipmi",
      "disk": "40",
      "arch": "x86_64",
      "cpu": "1",
      "pm_user": "admin"
    },
    {
      "pm_password": "testpass",
      "memory": "24",
      "pm_addr": "10.100.0.12",
      "mac": [
        "2c:c2:60:51:b7:fb"
      ],
      "pm_type": "ipmi",
      "disk": "40",
      "arch": "x86_64",
      "cpu": "1",
      "pm_user": "admin"
    }
  ],
  {
```

```
"pm_password": "testpass",
"memory": "24",
"pm_addr": "10.100.0.13",
"mac": [
  "2c:c2:60:76:ce:a5"
],
"pm_type": "ipmi",
"disk": "40",
"arch": "x86_64",
"cpu": "1",
"pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.51",
  "mac": [
    "2c:c2:60:08:b1:e2"
  ],
  "pm_type": "ipmi",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "1",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.52",
  "mac": [
    "2c:c2:60:20:a1:9e"
  ],
  "pm_type": "ipmi",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "1",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.53",
  "mac": [
    "2c:c2:60:58:10:33"
  ],
  "pm_type": "ipmi",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "1",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.101",
  "mac": [
```



```

    "2c:c2:60:31:a9:55"
  ],
  "pm_type": "ipmi",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "2",
  "pm_user": "admin"
},
{
  "pm_password": "testpass",
  "memory": "24",
  "pm_addr": "10.100.0.102",
  "mac": [
    "2c:c2:60:0d:e7:d1"
  ],
  "pm_type": "ipmi",
  "disk": "40",
  "arch": "x86_64",
  "cpu": "2",
  "pm_user": "admin"
}
],
"overcloud": {"password": "7adbbbeedc5b7a07ba1917e1b3b228334f9a2d4e",
"endpoint": "http://192.168.1.150:5000/v2.0/"
}
}

```

undercloud.conf

```

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

```

network-environment.yaml

```

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

```

```

storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/ceph-
storage.yaml

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ExternalNetCidr: 192.168.1.0/24
  InternalApiAllocationPools: [{start: 172.16.0.10, end: 172.16.0.200}]
  TenantAllocationPools: [{start: 172.17.0.10, end: 172.17.0.200}]
  StorageAllocationPools: [{start: 172.18.0.10, end: 172.18.0.200}]
  StorageMgmtAllocationPools: [{start: 172.19.0.10, end: 172.19.0.200}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{start: 192.168.1.150, end: 192.168.1.199}]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ExternalNetworkVlanID: 100
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.1.1
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: ""
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=active-backup lacp=off other_config:bond-miimon-interval=100"

```

2.4. 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)

此文件标识了 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
```

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
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

- [将 Red Hat Ceph Storage 和 Red Hat OpenStack Platform 与 director 一起部署](#)
- [第 1 章 Red Hat OpenStack Platform 高可用性概述和计划](#)

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、RabbitMQ、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。

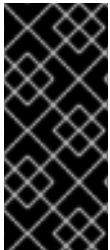
Pacemaker 管理以下资源类型：

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 HAProxy 等简单 **systemd** 服务到复杂的服务，如 Galera，这需要控制并在不同节点上设置服务状态的特定资源代理。



重要

- 您不能使用 **podman** 或 **systemctl** 管理捆绑包或容器。您可以使用 `pcs` 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，Podman 会将 **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. 简单捆绑包资源 (简单捆绑包)

简单的 Bundle Set 资源或简单捆绑包是一组您要在 Controller 节点上部署的相同 Pacemaker 服务的容器。

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 分配给服务的名称。
- 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。

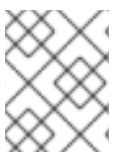
以下示例显示了 `haproxy-bundle` 简单捆绑包的设置：

```
$ 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)
```

示例显示了捆绑包中容器的以下信息：

- **镜像**：容器使用的镜像，它引用 undercloud 的本地 registry。
- **Network**: Container Network type，本例中为 "host"。
- **选项**：容器的特定选项。
- **副本**：指示容器必须在集群中运行多少个副本。每个捆绑包包含三个容器，每个 Controller 节点对应一个容器。
- **run-command**: 用于生成容器的 System 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平衡流量来提供高可用性服务，但您可以通过将其作为 Pacemaker 捆绑包服务进行管理，来将 HAProxy 配置为高度可用的服务。

3.1.2. 复杂的 Bundle Set 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，这些服务可以根据它们运行的 Controller 节点有不同的状态。

这个示例显示 `pcs status` 命令的输出中的复杂捆绑包列表：

```
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，类似于简单的捆绑包。
- Galera：所有三个 Controller 节点都作为 Galera master 运行，在同一约束下。
- Redis：**overcloud-controller-0** 容器作为 master 运行，而其他两个 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
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，与简单捆绑包不同，**galera-bundle** 资源包含明确的资源配置，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“在高可用性集群中查看虚拟 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，并查看有关活跃和运行的资源数量的信息。

先决条件

- 部署并运行高可用性。

流程

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

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

2. 运行 **pcs status** 命令：

```
[tripleo-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集群的以下信息：

- **集群名称**：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。

- **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. 在高可用性集群中检查捆绑包状态

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

- 部署并运行高可用性。

流程

使用以下选项之一 :

- 登录 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. 在高可用性集群中查看虚拟 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

- 部署并运行高可用性。

流程

- 以 **tripleo-admin** 用户身份登录任何 Controller 节点。

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

- 使用以下选项之一：

- 使用 **--full** 选项运行 **pcs resource show** 命令显示所有使用虚拟 IP 的资源：

```
$ 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 地址最初附加到特定的 Controller 节点。例如，**overcloud-controller-0** 上已启动 **192.168.1.150**。但是，如果该 Controller 节点失败，IP 地址将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	dhcp_start 和 dhcp_end 范围的一部分在 undercloud.conf 文件中设置为 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	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
172.19.0.10	提供对存储管理的访问	network-environment.yaml 文件中的 StorageMgmtAllocationPools

- 运行 **pcs resource show** 命令并使用使用该 VIP 的资源的名称来查看特定的 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. 在高可用性集群中查看虚拟 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

- 部署并运行高可用性。

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

```

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

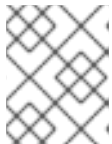
```
$ sudo netstat -tupln | grep "192.168.1.150.haproxy"
```

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**）也通过 **192.168.1.150** 提供。这些进程包括 **sshd**、**mysqld**、**dhclient**、**ntpd**。

- 打开 HA 服务的配置文件，查看默认端口号分配及其监听的服务，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，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，所有其他服务都将在回退 Controller 节点上运行。

3.6. 检查隔离代理和 PACEMAKER 守护进程状态

您可以在 Pacemaker 运行的任何节点中检查隔离代理的状态以及 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

- 部署并运行高可用性。

流程

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

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

2. 运行 **pcs status** 命令：

```
[tripleo-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 节点当前都在线。
- **守护进程状态**：显示三个 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且已启用。

3.7. 其他资源

- [配置和管理高可用性集群](#)

第 4 章 使用 STONITH 隔离 CONTROLLER 节点

隔离是隔离故障节点以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 的进程隔离故障节点。STONITH 是 "Shoot the other node 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 节点无法进行健康检查，则作为 Pacemaker 指定协调器(DC)的 Controller 节点使用 Pacemaker **stonith** 服务隔离受影响的 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，您可以根据环境需求选择隔离代理。要更改隔离代理，您必须在 **fencing.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（基于存储的 Death）守护进程与 Pacemaker 和 watchdog 设备集成，以安排节点在触发隔离时可靠地关闭，以防传统隔离机制不可用。



重要

- 使用 **pacemaker_remote** 的远程裸机或虚拟机节点不支持 SBD 隔离，因此如果部署使用了 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 隔离只支持兼容的 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。如果选择这个代理，请确保您有足够的磁盘空间来存储转储文件。

除了 IPMI、**fence_mvapich** 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，请确保分配足够时间以便第一个代理在第二个代理启动下一个任务前完成任务。



重要

- RHOSP director 仅支持 **fence_kdump** STONITH 代理的配置，而不是配置隔离代理所依赖的完整 **kdump** 服务。有关配置 **kdump** 服务的详情，请参考 [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** 或 **ovs_bonds** 网络设备，则 **fence_kdump** 不被支持。要启用 **fence_kdump**，必须将网络设备改为 **linux_bond** 或 **linux_bridge**。

Redfish

在带有支持 DMTF Redfish API 的服务器中使用。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](#)。

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，您可以使用 **fence_kdump** 配置 IPMI 隔离。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 “测试 overcloud 上的隔离”](#)
- [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，首先检查 STONITH 和 Pacemaker 的状态并配置 **fencing.yaml** 文件。然后，部署 overcloud 并配置其他参数。最后，测试 overcloud 上是否正确部署了隔离。

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”](#)。
- 确保您可以访问在 director 中注册节点时创建的 **nodes.json** 文件。此文件是您部署期间生成的 **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的 MAC 地址。有关更多信息，请参阅 [Overcloud 注册节点](#)。

流程

1. 以 **tripleo-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 **fencing.yaml** 环境文件：

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



注意

此命令将 **ilo** 和 **drac** 电源管理详细信息转换为 IPMI 等效的详细信息。

- 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 **fencing.yaml** 文件。

5. 仅 SBD 隔离：在 **fencing.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. 仅限多层隔离：将特定于级别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>` 和 `<value>` 替换为隔离代理需要的实际参数和值。

7. 运行 **overcloud deploy** 命令，并包含 **fence.yaml** 文件以及与您的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 tripleo-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

您不能在其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>

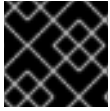
```

其它资源

- [第 4.3 节 “测试 overcloud 上的隔离”](#)
- [第 4.5 节 “隔离参数”](#)
- [探索 RHEL 高可用性的组件 - sbd 和 fence_sbd](#)

4.3. 测试 OVERCLOUD 上的隔离

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



重要

此流程降级到 Controller 节点的所有连接，这会导致节点重启。

先决条件

- 隔离部署在 overcloud 上并运行。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，再提供凭据文件：

```
$ source stackrc
$ openstack server list | grep controller
$ ssh tripleo-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 节点中找到 Pacemaker 日志文件中的隔离事件：

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

如果 STONITH 服务在 Controller 上执行隔离操作，则日志文件会显示一个隔离事件。

4. 等待几分钟，然后运行 **pcs status** 命令来验证重启的 Controller 节点是否再次在集群中运行。如果您可以看到您在输出中重启的 Controller 节点，则隔离功能可以正常工作。

4.4. 查看 STONITH 设备信息

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

先决条件

- 隔离部署在 overcloud 上并运行。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：

- 隔离设备使用的 IPMI 电源管理服务根据需要用于打开和关闭计算机，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**。
- 用于登录到节点的密码，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

在 overcloud 上部署隔离时，您可以使用所需的参数生成 **fencing.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：

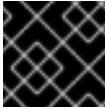
EnableFencing

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：

- **agent**: 隔离代理名称。
- **host_mac** : 部署接口小写或服务器上的任何其他网络接口的 mac 地址。您可以使用它作为隔离设备的唯一标识符。



重要

不要使用 IPMI 接口的 MAC 地址。

- **参数** : 隔离设备参数列表。

隔离设备参数

列出隔离设备参数。本例显示了 IPMI 隔离代理的参数：

- **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 设备上的特权级别
- **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 负载均衡流量

HAProxy 服务提供高可用性集群中 Controller 节点的流量负载均衡，以及日志记录和示例配置。**haproxy** 软件包包含 **haproxy** 守护进程，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 的工作原理

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 节点上的专用容器中运行。

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，您可以查看以下属性：

- **listen**: 正在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。
- **server** : 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，可从 overcloud 外部访问 API 网络。

- **8776** : OpenStack Block Storage (cinder)服务侦听的端口。
- **Server** : Controller 节点名称和 IP 地址。HAProxy 可以将对这些 IP 地址的请求定向到 **服务器** 输出中列出的其中一个 Controller 节点。
- **httpchk** : 启用 Controller 节点服务器上的健康检查。
- **回退到 5** : 用于确定服务离线的健康检查数量。
- **inter 2000** : 以毫秒为单位, 两个连续健康检查之间的间隔。
- **增长 2** : 用于确定服务是否正在运行的健康检查数量。

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 还在所有 HA 部署上启用 HAProxy Stats 或 统计数据。通过此功能, 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

- 部署并运行高可用性。

流程

1. 在安装了 HAProxy 的任何 Controller 节点上, 打开 **/var/lib/config-data/puppet-generated/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**, 再输入 **stats auth** 行中的凭据来查看 HAProxy Stats 页面。

5.3. 其他资源

- [HAProxy 1.8 文档](#)
- [如何验证我的 haproxy.cfg 是否已正确配置为负载均衡 openstack 服务?](#)

第 6 章 将 OVERCLOUD 配置为使用外部负载均衡器

在 Red Hat OpenStack Platform (RHOSP) 中，overcloud 将多个 Controller 节点用作高可用性集群，以确保 OpenStack 服务的最大操作性能。集群还为 OpenStack 服务提供负载均衡，该服务平均将流量分发到 Controller 节点，并减少每个节点的服务器过载。

默认情况下，overcloud 使用名为 HAProxy 的开源工具来管理负载平衡。HAProxy 负载均衡流量到运行 OpenStack 服务的 Controller 节点。**haproxy** 软件包包含侦听传入流量的 **haproxy** 守护进程，包括日志记录功能和示例配置。

overcloud 还使用高可用性资源管理器 Pacemaker 将 HAProxy 作为高可用性服务控制。这意味着 HAProxy 在每个 Controller 节点上运行，并根据您在每个配置中定义的一组规则分发流量。

您还可以使用外部负载均衡器来执行此分发。例如，您的组织可能会使用基于硬件的专用负载均衡器来处理到 Controller 节点的流量分布。要定义外部负载均衡器和 overcloud 创建的配置，您需要执行以下进程：

1. 安装和配置外部负载均衡器。
2. 使用 heat 模板参数配置和部署 overcloud，以将 overcloud 与外部负载均衡器集成。这需要负载均衡器和潜在的节点的 IP 地址。

在将 overcloud 配置为使用外部负载均衡器前，请确保在 overcloud 上部署并运行高可用性。

6.1. 为外部负载均衡器准备您的环境

要为外部负载均衡器准备您的环境，首先创建一个节点定义模板，并使用 director 注册空白节点。然后，检查所有节点的硬件，并手动标记节点到配置集。

使用以下工作流准备您的环境：

- 创建节点定义模板，并使用 Red Hat OpenStack Platform director 注册空白节点。**instackenv.json** 的节点定义模板是一个 JSON 格式文件，其中包含用于注册节点的硬件和电源管理详情。
- 检查所有节点的硬件。这样可确保所有节点都处于 manageable 状态。
- 手动将节点标记为配置集。这些配置集标签与节点与类型匹配。然后，类别分配到一个部署角色。

流程

1. 以 **stack** 用户身份登录 director 主机，再提供 director 凭证：

```
$ source ~/stackrc
```

2. 在 **stackenv.json** 中创建节点定义模板，并根据您的环境编辑以下示例：

```
{
  "nodes":[
    {
      "mac":[
        "bb:bb:bb:bb:bb:bb"
      ],
      "cpu":"4",
```

```

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

```

3. 将文件保存到 **stack** 用户的主目录 `/home/stack/instackenv.json` 中，然后将其导入到 director，然后将节点注册到 director：

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

4. 将内核和 ramdisk 镜像分配给所有节点：

```
$ openstack overcloud node configure
```

5. 检查每个节点的硬件属性：

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



重要

节点必须处于可管理的状态。确保此进程完成运行。它可能需要 15 分钟来检查这些裸机节点。

6. 获取节点列表以识别它们的 UUID：

```
$ openstack baremetal node list
```

7. 通过在每个节点的 **properties/capabilities** 参数中添加 profile 选项，手动将每个节点标记到特定的配置集。例如，要标记三个节点以使用 Controller 配置集和一个节点来使用 Compute 配置集，请使用以下命令：

```
$ openstack baremetal node set 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 --property capabilities='profile:control,boot_option:local'
$ openstack baremetal node set 6faba1a9-e2d8-4b7c-95a2-c7fbd12129a --property capabilities='profile:control,boot_option:local'
$ openstack baremetal node set 5e3b2f50-fcd9-4404-b0a2-59d79924b38e --property capabilities='profile:control,boot_option:local'
$ openstack baremetal node set 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13 --property capabilities='profile:compute,boot_option:local'
```

profile:compute 和 **profile:control** 选项将节点标记为每个对应的配置集。

其他资源

- [规划您的 overcloud](#)

6.2. 为外部负载均衡器配置 OVERCLOUD 网络

要为 overcloud 配置网络，请隔离您的服务以使用特定的网络流量，然后为本地环境配置网络环境文件。此文件是描述 overcloud 网络环境的 heat 环境文件，指向网络接口配置模板，并为您的网络和 IP 地址范围定义子网和 VLAN。

流程

1. 要为各个角色配置节点接口，请自定义以下网络接口模板：

- 要为每个角色配置带有 VLAN 的单个 NIC，请使用以下目录中的示例模板：

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

- 要为每个角色配置绑定 NIC，请使用以下目录中的示例模板：

```

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

```

2. 通过从 `/home/stack/network-environment.yaml` 中复制文件并根据您的环境编辑内容来创建网络环境文件。

其他资源

- [基本网络隔离](#)
- [自定义可组合网络](#)
- [自定义网络接口模板](#)
- [Overcloud 网络](#)

6.3. 创建外部负载均衡器环境文件

要使用外部负载均衡器部署 overcloud，请使用所需的配置创建新的环境文件。在本示例文件中，外部负载均衡器上配置多个虚拟 IP、每个隔离网络上的一个虚拟 IP，另一个用于 Redis 服务，再启动 overcloud 部署。如果 overcloud 节点 NIC 配置支持配置，则一些虚拟 IP 可以相同。

流程

- 使用以下示例环境文件 `external-lb.yaml` 创建环境文件，并根据您的环境编辑内容。

```

parameter_defaults:
  ControlFixedIPs: [{'ip_address':'192.0.2.250'}]
  PublicVirtualFixedIPs: [{'ip_address':'172.16.23.250'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.20.250'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.16.21.250'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.16.19.250'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.16.20.249'}]
  # IPs assignments for the Overcloud Controller nodes. Ensure these IPs are from each
  # respective allocation pools defined in the network environment file.
  ControllerIPs:
    external:
      - 172.16.23.150
      - 172.16.23.151
      - 172.16.23.152
    internal_api:
      - 172.16.20.150
      - 172.16.20.151
      - 172.16.20.152
    storage:
      - 172.16.21.150
      - 172.16.21.151
      - 172.16.21.152
    storage_mgmt:
      - 172.16.19.150
      - 172.16.19.151
      - 172.16.19.152
    tenant:
      - 172.16.22.150
      - 172.16.22.151
      - 172.16.22.152

```

```

# CIDRs
external_cidr: "24"
internal_api_cidr: "24"
storage_cidr: "24"
storage_mgmt_cidr: "24"
tenant_cidr: "24"
RedisPassword: p@55w0rd!
ServiceNetMap:
  NeutronTenantNetwork: tenant
  CeilometerApiNetwork: internal_api
  AodhApiNetwork: internal_api
  GnocchiApiNetwork: internal_api
  MongoDBNetwork: internal_api
  CinderApiNetwork: internal_api
  CinderIscsiNetwork: storage
  GlanceApiNetwork: storage
  GlanceRegistryNetwork: internal_api
  KeystoneAdminApiNetwork: internal_api
  KeystonePublicApiNetwork: internal_api
  NeutronApiNetwork: internal_api
  HeatApiNetwork: internal_api
  NovaApiNetwork: internal_api
  NovaMetadataNetwork: internal_api
  NovaVncProxyNetwork: internal_api
  SwiftMgmtNetwork: storage_mgmt
  SwiftProxyNetwork: storage
  HorizonNetwork: internal_api
  MemcachedNetwork: internal_api
  RabbitMqNetwork: internal_api
  RedisNetwork: internal_api
  MysqlNetwork: internal_api
  CephClusterNetwork: storage_mgmt
  CephPublicNetwork: storage
  ControllerHostnameResolveNetwork: internal_api
  ComputeHostnameResolveNetwork: internal_api
  BlockStorageHostnameResolveNetwork: internal_api
  ObjectStorageHostnameResolveNetwork: internal_api
  CephStorageHostnameResolveNetwork: storage

```



注意

- **parameter_defaults** 部分包含每个网络的 VIP 和 IP 分配。这些设置必须与负载均衡器中每个服务相同的 IP 配置匹配。
- **parameter_defaults** 部分定义 Redis 服务的管理密码(RedisPassword)，其中包含 **ServiceNetMap** 参数，该参数将每个 OpenStack 服务映射到特定的网络。负载均衡配置需要重新映射此服务。

6.4. 为外部负载均衡配置 SSL

要为外部负载均衡器配置加密端点，请创建额外的环境文件，使 SSL 访问端点，然后在外部负载均衡服务器上安装 SSL 证书和密钥的副本。默认情况下，overcloud 使用未加密的端点服务。

先决条件

- 如果您使用 IP 地址或域名访问公共端点，请选择以下环境文件之一包括在 overcloud 部署中：
 - 要使用域名服务(DNS)访问公共端点，请使用文件 `/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-dns.yaml`。
 - 要使用 IP 地址访问公共端点，请使用文件 `/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-ip.yaml`。

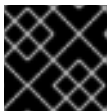
流程

1. 如果您使用自签名证书，或者证书签名者不在 overcloud 镜像的默认信任存储中，请通过从 heat 模板集合中复制 `inject-trust-anchor.yaml` 环境文件来将证书注入 overcloud 镜像中：

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

2. 在文本编辑器中打开该文件，并将 root 证书颁发机构文件的内容复制到 `SSLRootCertificate` 参数：

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgx CzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mIBBTFTCMBAj6rtLBKLaQblxEplzrgvp
    -----END CERTIFICATE-----
```



重要

证书颁发机构内容为所有新行需要相同的缩进级别。

3. 将 `OS::TripleO::NodeTLSCAData:` 参数的资源 URL 更改为绝对 URL：

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

4. 可选：如果您使用 DNS 主机名通过 SSL/TLS 访问 overcloud，请创建一个新的环境文件 `~/templates/cloudname.yaml`，并定义 overcloud 端点的主机名：

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

将 `overcloud.example.com` 替换为 overcloud 端点的 DNS 主机名。

6.5. 使用外部负载均衡器部署 OVERCLOUD

要部署使用外部负载均衡器的 overcloud，请运行 `openstack overcloud deploy`，并包含外部负载均衡器的额外环境文件和配置文件。

先决条件

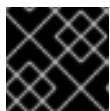
- 为外部负载均衡器准备环境。有关如何准备您的环境的更多信息，请参阅 [第 6.1 节 “为外部负载均衡器准备您的环境”](#)
- 为外部负载均衡器配置了 overcloud 网络。有关如何配置网络的详情，请参考 [第 6.2 节 “为外部负载均衡器配置 overcloud 网络”](#)
- 已准备好外部负载均衡器环境文件。有关如何创建环境文件的详情，请参考 [第 6.3 节 “创建外部负载均衡器环境文件”](#)
- 为外部负载平衡配置了 SSL。有关如何为外部负载平衡配置 SSL 的详情，请参考 [第 6.4 节 “为外部负载平衡配置 SSL”](#)

流程

1. 使用外部负载均衡器的所有环境和配置文件部署 overcloud :

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

将尖括号 <> 中的值替换为您为环境定义的文件路径。



重要

您必须按照本例中列出的顺序将网络环境文件添加到命令中。

这个命令包括以下环境文件：

- **network-isolation.yaml** : 网络隔离配置文件。
- **network-environment.yaml** : 网络配置文件。
- **external-loadbalancer-vip.yaml** : 外部负载均衡虚拟 IP 地址配置文件。
- **external-lb.yaml** : 外部负载均衡器配置文件。您还可以为此文件设置以下选项，并调整您的环境的值：
 - **--control-scale 3**: 将 Controller 节点扩展为 3。
 - **--compute-scale 3**: 将 Compute 节点扩展为 3。
 - **--control-flavor control** : 将特定类别用于 Controller 节点。
 - **--compute-flavor compute** : 将特定类别用于 Compute 节点。
- SSL/TLS 环境文件：

- **SSL/TLS 端点环境文件**：定义如何连接到 public endpointst 的环境文件。使用 **tls-endpoints-public-dns.yaml** 或 **tls-endpoints-public-ip.yaml**。
- （可选） **DNS 主机名环境文件**：设置 DNS 主机名的环境文件。
- **根证书注入环境文件**：用于注入 root 证书颁发机构的环境文件。

在 overcloud 部署期间，Red Hat OpenStack Platform director 会置备您的节点。完成此过程需要一些时间。

2. 要查看 overcloud 部署的状态，请输入以下命令：

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

6.6. 其他资源

- [使用 HAProxy 负载均衡流量](#).

第 7 章 示例配置：带有外部 HAPROXY 负载均衡器的 OVERCLOUD

此示例配置显示使用联合 HAProxy 服务器提供外部负载平衡的 overcloud。您可以根据您的环境要求选择不同的外部负载均衡器。

示例配置包括以下元素：

- 运行 HAProxy 的外部负载均衡服务器。
- 一个 Red Hat OpenStack Platform (RHOSP) director 节点。
- 一个 overcloud，它由高可用性集群中的 3 个 Controller 节点和 1 个 Compute 节点组成。
- 使用 VLAN 进行网络隔离。

这个示例为每个网络使用以下 IP 地址分配：

- 内部 API: **172.16.20.0/24**
- 租户：**172.16.22.0/24**
- Storage: **172.16.21.0/24**
- 存储管理：**172.16.19.0/24**
- 外部：**172.16.23.0/24**

这些 IP 范围包括 Controller 节点和负载均衡器绑定到 OpenStack 服务的虚拟 IP 的 IP 分配。

7.1. HAPROXY 配置文件示例

示例文件显示了内部 HAProxy 配置参数。您可以使用示例配置参数作为配置外部负载均衡器的基础。

HAProxy 配置文件包含以下部分：

- 全局配置
- 默认配置
- 服务配置

director 在每个 Controller 节点上的 `/etc/haproxy/haproxy.conf` 文件中为非容器化环境提供此配置，并在 `/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg` 文件中为容器化环境提供。



注意

除了全局、默认和服务参数外，还必须配置其他 HAProxy 参数。有关 HAProxy 参数的更多信息，请参阅位于 Controller 节点上的 `/usr/share/doc/haproxyFE/configuration.txt` 中的 *HAProxy 配置手册*，或在安装了 `haproxy` 软件包的任何系统中。

HAProxy 配置文件示例

global

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

defaults

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

listen aodh

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

listen ceilometer

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

listen cinder

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

listen glance_api

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

listen glance_registry

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

listen gnocchi

```
bind 172.16.23.250:8041
bind 172.16.21.250:8041
```

```
mode http
server overcloud-controller-0 172.16.20.150:8041 check fall 5 inter 2000 rise 2
server overcloud-controller-1 172.16.20.151:8041 check fall 5 inter 2000 rise 2
server overcloud-controller-2 172.16.20.152:8041 check fall 5 inter 2000 rise 2
```

```
listen heat_api
```

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

```
listen heat_cfn
```

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

```
listen heat_cloudwatch
```

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

```
listen horizon
```

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

```
listen keystone_admin
```

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

```
listen keystone_admin_ssh
```

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

```
listen keystone_public
```

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

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

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

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

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

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

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

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

```

server overcloud-controller-2 172.16.20.152:8778 check fall 5 inter 2000 rise 2

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

listen redis
bind 172.16.20.249:6379
balance first
option tcp-check
tcp-check send AUTH\r\n p@55w0rd!\r\n
tcp-check send PING\r\n
tcp-check expect string +PONG
tcp-check send info\r\n replication\r\n
tcp-check expect string role:master
tcp-check send QUIT\r\n
tcp-check expect string +OK
server overcloud-controller-0 172.16.20.150:6379 check fall 5 inter 2000 rise 2
server overcloud-controller-1 172.16.20.151:6379 check fall 5 inter 2000 rise 2
server overcloud-controller-2 172.16.20.152:6379 check fall 5 inter 2000 rise 2

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

```

7.1.1. 全局配置参数：HAProxy 配置文件示例

全局配置参数部分为负载均衡器定义一组进程范围参数。您可以使用配置文件中的示例参数来配置外部负载均衡器。根据您的环境调整参数值。

全局配置参数

```

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

```

示例显示了以下参数：

- **守护进程**：作为后台进程运行。
- **用户 haproxy 和组 haproxy**：定义拥有进程的 Linux 用户和组。
- **log**：定义要使用的 syslog 服务器。
- **maxconn**：设置进程的并发连接的最大数量。

- **pidfile** : 设置用于进程 ID 的文件。

7.1.2. 默认值配置参数：HAProxy 配置文件示例

默认值配置参数部分定义在运行外部负载均衡器服务时要使用的一组默认值。您可以使用配置文件中的示例参数来配置外部负载均衡器。根据您的环境调整参数值。

默认值配置参数

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

示例显示了以下参数：

- **日志** : 启用该服务的日志记录。**global** 值表示日志记录功能使用 **global** 部分中的 **log** 参数。
- **模式** : 定义要使用的协议。在这种情况下，默认为 TCP。
- **retries** : 在报告连接失败前设置服务器执行的重试次数。
- **timeout** : 设置等待特定功能的最长时间。例如，**超时 http-request** 设置等待完整 HTTP 请求的最长时间。

7.1.3. 服务级别配置参数：HAProxy 配置文件示例

服务级别配置参数部分定义在将流量负载均衡到特定 Red Hat OpenStack Platform (RHOSP) 服务时使用的一组参数。您可以使用配置文件中的示例参数来配置外部负载均衡器。根据您的环境调整参数值，并为您要负载均衡的每个服务复制部分。

服务级别配置参数

此示例显示 **ceilometer** 服务的配置参数。

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

要负载均衡的每个服务都必须与配置文件中的一个部分对应。每个服务配置包括以下参数：

- **listen**: 侦听请求的服务名称。
- **绑定** : 服务侦听的 IP 地址和 TCP 端口号。每个服务绑定了代表不同网络流量类型的不同地址。
- **server** : 提供服务的每个服务器的名称、服务器 IP 地址和侦听端口，以及连接参数：

- **检查**：（可选）启用健康检查。
- **回退 5**：（可选）在五个失败的健康检查后，该服务被视为离线。
- **inter 2000**：（可选）两个连续健康检查设置为 2000 毫秒或 2 秒之间的间隔。
- **增长 2**：（可选）在两个成功健康检查后，该服务被视为可操作。

在 **ceilometer** 示例中，服务标识了提供 **ceilometer** 服务的 IP 地址和端口 **172.16.20.2500:8777** 和 **172.16.23.250:8777**。HAProxy 将对这些地址的请求定向到 **overcloud-controller-0** (172.16.20.150:8777)、**overcloud-controller-1** (172.16.20.151:8777) 或 **overcloud-controller-2** (172.16.0.152:8777)。

其他资源

- [第 7.2 节 “使用负载均衡的服务的配置参数”](#)

7.2. 使用负载均衡的服务的配置参数

对于使用负载均衡的 overcloud 中的每个服务，请使用以下示例作为配置外部负载均衡器的指南。根据您的环境调整参数值，并为您要负载均衡的每个服务复制部分。



注意

大多数服务使用默认的健康检查配置：

- 两个连续健康检查之间的间隔设置为 2000 毫秒，或 2 秒。
- 两个成功健康检查后，服务器将被视为可操作。
- 在五个健康检查失败后，该服务被视为离线。

每个服务都指示该服务的 **Other information** 部分中的默认健康检查或附加选项。

aodh

端口号：8042

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

ceilometer

端口号：8777

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

cinder

端口号：8776

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

glance_api

端口号：9292

绑定到：storage, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的存储

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

glance_registry

端口号：9191

绑定到：internal_api

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

gnocchi

端口号：8041

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

heat_api

端口号：8004

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。
- 此服务使用 HTTP 模式而不是默认的 TCP 模式。

HAProxy 示例：

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

heat_cfn

端口号：8000

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

heat_cloudwatch

端口号：8003

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

horizon

端口号：80

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。
- 此服务使用 HTTP 模式而不是默认的 TCP 模式。
- 此服务使用基于 Cookie 的持久性来与 UI 交互。

HAProxy 示例：

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

keystone_admin

端口号：35357

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

keystone_admin_ssh

端口号：22

绑定到：internal_api

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

keystone_public

端口号：5000

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

mysql

端口号：3306

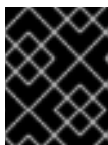
绑定到：internal_api

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。但是，健康检查使用端口 9200。
- 此服务正一个时间点上只会负载均衡一台服务器。

- 只有在所有其他非备份服务器都不可用时，每个服务器才会在负载均衡中使用。
- 如果服务器离线，则所有连接都会立即终止。
- 您必须在两端启用 TCP keepalive 数据包发送。
- 您必须启用 HTTP 协议来检查服务器健康状况。
- 您可以配置粘性表来存储 IP 地址，以帮助保持持久性。



重要

mysql 服务使用 Galera 提供高度可用的数据库集群。Galera 支持主动-主动配置，但为了避免锁定竞争，您必须使用负载均衡器强制执行的主动 - 被动配置。

HAProxy 示例：

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

neutron

端口号：9696

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

nova_ec2

端口号：8773

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

nova_metadata

端口号：8775

绑定到：internal_api

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

nova_novncproxy

端口号：6080

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。
- 默认负载均衡方法是 round-robin。但是，对于此服务，请使用 **源** 方法。这个方法对源 IP 地址进行哈希处理，并使用正在运行的服务器的总权重来划分它。这个方法还指定接收请求的服务器，并确保相同的客户端 IP 地址始终到达同一服务器，除非服务器停机或启动。如果由于运行服务器

数量的变化导致哈希结果发生变化，负载均衡器会将客户端重定向到不同的服务器。

HAProxy 示例：

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

nova_osapi

端口号：8774

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

nova_placement

端口号：8778

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

panko

端口号：8779

绑定到：internal_api, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

redis

端口号：6379

绑定到：internal_api (redis 服务 IP)

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的 internal_api

其他信息：

- 每个目标服务器都使用默认健康检查。
- 使用 **tcp-check** send/expect 序列执行健康检查。要发送的字符串是 **info\ replication\r\n**，响应是 **role:master**。
- Redis 服务使用密码进行身份验证。例如，HAProxy 配置使用带有 AUTH 方法和 Redis 管理密码的 **tcp-check**。director 通常会生成一个随机密码，但您可以定义自定义 Redis 密码。
- 默认平衡方法是 **round-robin**。但是，对于此服务，请使用 **第一个** 方法。这样可确保第一个具有可用连接插槽的服务器接收连接。

HAProxy 示例：

```
listen redis
bind 172.16.20.249:6379 transparent
balance first
option tcp-check
tcp-check send AUTH\ p@55w0rd!\r\n
tcp-check send PING\r\n
tcp-check expect string +PONG
tcp-check send info\ replication\r\n
tcp-check expect string role:master
tcp-check send QUIT\r\n
tcp-check expect string +OK
```

```
server overcloud-controller-0 172.16.20.150:6379 check fall 5 inter 2000 rise 2
server overcloud-controller-1 172.16.20.151:6379 check fall 5 inter 2000 rise 2
server overcloud-controller-2 172.16.20.152:6379 check fall 5 inter 2000 rise 2
```

swift_proxy_server

端口号：8080

绑定到：storage, external

目标网络或服务器：overcloud-controller-0、overcloud-controller-1 和 overcloud-controller-2 上的存储

其他信息：

- 每个目标服务器都使用默认健康检查。

HAProxy 示例：

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

第 8 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，该资源管理数据库主/从状态。您可以使用 Galera 测试并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。
- 节点可以从集群中接收查询和写入命令。
- 节点连接到集群中的其他节点。
- 节点将写入集复制到本地数据库中的表。

8.1. 在 MARIADB 集群中验证主机名解析

要对 MariaDB Galera 集群进行故障排除，首先消除任何主机名解析问题，然后在每个 Controller 节点的数据库中检查 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（如错误配置或失败 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，通过运行 **hieria** 命令获取 MariaDB 数据库 root 密码。

```
$ sudo hieria -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-PASS
WORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_%';"
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| wsrep_applier_thread_count | 1 |
| wsrep_apply_oooe   | 0.018672 |
| wsrep_apply_ool    | 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数量，以验证 MariaDB Galera 集群的健康和完整性。

8.2. 检查 MARIADB 集群完整性

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，将 `<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 8.1. 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。如果该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	集群更改的总数	决定集群是否被分成多个组件或分区。分区通常是由网络故障造成的。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)。

变量	概述	描述
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，则 wsrep_cluster_status 值为 Primary 。任何其他值表示节点是无法正常工作的分区的一部分。

8.3. 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，请通过检查特定的 **wsrep** 数据库变量来检查节点的完整性。

流程

- 运行以下命令，将 `<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 8.2. 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写集。如果是，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，则 wsrep_connected 为 ON 。
wsrep_local_state_comment	节点状态	总结了节点状态。如果节点可以写入集群，则 wsrep_local_state_comment 的典型值可以是 Joining , Waiting on SST , Joined , Synced , 或 Donor 。 如果节点是非正常运行的组件的一部分，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 8.2 节“检查 MariaDB 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。

8.4. 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** 数据库变量在集群的复制吞吐量上运行基准测试。

每次查询这些变量时，**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并分析差异。这些差异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，流控制会暂停复制，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，将 `<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 8.3. 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接收写入队列的平均大小。	高于 0.0 的值表示节点无法像接收写集那样快速应用写集，这会触发复制节流。检查 wsrep_local_recv_queue_min 和 wsrep_local_recv_queue_max ，以了解这个基准的详情。
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 的值高于 0.0，您可以检查这些变量以确定队列大小的范围。

变量	概述	使用方法
wsrep_flow_control_paused	流控制在最后一次查询后暂停节点的部分。	<p>高于 0.0 的值表示 Flow Control 暂停节点。要确定暂停的持续时间，请将 wsrep_flow_control_paused 值与查询之间的秒数相乘。最佳值尽可能接近 0.0。</p> <p>例如：</p> <ul style="list-style-type: none"> 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 Flow Control 会暂停整个一分钟的节点。
wsrep_cert_deps_distance	可以并行应用的最低序列号和最高序列号(seqno)值的平均区别	如果出现节流和暂停，这个变量表示平均可以并行应用多少写集。将值与 wsrep_slave_threads 变量进行比较，以查看实际可以同时应用多少写集。
wsrep_slave_threads	可同时应用的线程数量	<p>您可以增加此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 的值不能大于节点的 CPU 内核数。</p> <p>例如，如果 wsrep_cert_deps_distance 的值是 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集量。</p> <p>如果有问题的节点已经有一个最佳 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。</p>

8.5. 其他资源

- [什么是 MariaDB Galera 集群？](#)

第 9 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。根据您的部署，资源失败有很多，您必须调查资源以确定如何解决问题。

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。您还可以检查被隔离的 Controller 节点，超过其他 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，这可能代表 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，并检查输出以了解特定 Pacemaker 资源故障的信息，或运行 **systemctl** 命令来调查非 Pacemaker 资源失败。

9.1. 在高可用性集群中查看资源限制

在调查资源问题前，您可以查看服务启动方式的限制，包括与每个资源所在的位置相关的约束，资源启动顺序，以及资源是否与另一个资源在一起。

流程

- 使用以下选项之一：
 - 要查看所有资源限制，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)

```

此输出显示以下主要约束类型：

位置约束

列出可分配给资源的位置：

- 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，这是使服务可访问所必需的。
- 第三个位置约束显示在每个 Controller 节点上禁用 **ipmilan** 资源。

排序约束

列出资源可以启动的顺序。本例显示了一个约束，其将虚拟 IP 地址资源 **IPAddr2** 设置为在 HAProxy 服务之前启动。



注意

排序限制只适用于 IP 地址资源和 HAProxy。systemd 管理所有其他资源，因为 Compute 等服务预期会中断依赖服务，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** 属性为 **true**，这意味着 **galera-bundle** 资源仅在这些节点上运行。

9.2. 检查 PACEMAKER 资源问题

要调查 Pacemaker 管理的失败资源，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 **openstack-cinder-volume** 资源的状态和日志事件。

先决条件

- 具有 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\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. 根据输出中的信息以及日志中的信息更正失败的资源。

- 运行 **pcs resource cleanup** 命令以重置状态以及资源的失败计数。

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

9.3. 调查 SYSTEMD 资源问题

要调查 systemd 管理的失败资源，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

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

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

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

- 查看资源的日志事件：

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

- 根据输出中的信息以及日志中的信息更正失败的资源。
- 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
0d6583beb20508e6baccdd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -u
0d6583beb20508e6baccdd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -r /usr/bin/runc
-b /var/lib/containers>

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 10 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** monitor 守护进程来管理 Ceph 集群。director 在所有 Controller 节点上部署守护进程。

10.1. 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 **service ceph status** 命令。

流程

- 登录 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"}
```

10.2. 检查 RED HAT CEPH 监控配置

要检查 Red Hat Ceph Storage 监控服务的配置，请登录到 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，并打开 **/etc/ceph/ceph.conf** 文件来查看监控配置参数：

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

这个示例显示以下信息：

- 所有三个 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，监控 Controller 节点的 IP 地址是 **172.18.0.15**、**172.18.0.16** 和 **172.18.0.17**。

10.3. 检查 RED HAT CEPH 节点状态

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

流程

- 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
```

本例输出显示 **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。

10.4. 其他资源

- [Red Hat Ceph 产品页面](#)