Red Hat OpenStack Platform 13

고가용성 배포 및 사용

Red Hat OpenStack Platform에서 고가용성 계획, 배포 및 관리

Last Updated: 2023-04-08

Red Hat OpenStack Platform 13 고가용성 배포 및 사용

Red Hat OpenStack Platform에서 고가용성 계획, 배포 및 관리

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

법적 공지

초록

To keep your OpenStack environment up and running efficiently, use the Red Hat OpenStack Platform director to create configurations that offer high availability and load-balancing across all major services in OpenStack.

차례

머리말

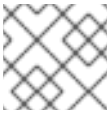
1장. 고가용성 서비스

RHOSP(Red Hat OpenStack Platform)는 HA(고가용성)를 구현하는 데 필요한 서비스를 제공하기 위해 여러 기술을 사용합니다.

서비스 유형

코어 컨테이너

핵심 컨테이너 서비스는 *Galera*,*RabbitMQ*,*Redis*, *HAProxy* 입니다. 이러한 서비스는 모든 컨트롤러 노드에서 실행되며 시작, 중지 및 재시작 작업을 위해 특정 관리 및 제약 조건이 필요합니다. Pacemaker를 사용하여 핵심 컨테이너 서비스를 시작, 관리 및 해결합니다.



참고

RHOSP는 MariaDB Galera 클러스터를 사용하여 데이터베이스 복제를 관리합니다.

active-passive

활성-수동 서비스는 한 번에 하나의 컨트롤러 노드에서 실행되며 **openstack-cinder-volume** 과 같은 서비스를 포함합니다. active-passive 서비스를 이동하려면 Pacemaker를 사용하여 올바른 stop-start 시퀀스를 따라야 합니다.

systemd 및 일반 컨테이너

systemd 및 일반 컨테이너 서비스는 서비스 중단을 수행할 수 있는 독립 서비스입니다. 따라서 Galera와 같은 고가용성 서비스를 다시 시작하면 **nova-api** 와 같은 다른 서비스를 수동으로 다시 시작할 필요가 없습니다. systemd 또는 Docker를 사용하여 systemd 및 일반 컨테이너 서비스를 직접 관리할 수 있습니다.

director와 함께 HA 배포를 오케스트레이션할 때 director는 템플릿 및 Puppet 모듈을 사용하여 모든 서비스가 올바르게 구성 및 시작되도록 합니다. 또한 HA 문제를 해결할 때 **docker** 명령 또는 **systemctl** 명령을 사용하여 HA 프레임워크의 서비스와 상호 작용해야 합니다.

서비스 모드

HA 서비스는 다음 모드 중 하나로 실행할 수 있습니다.

- **Active-active**: Pacemaker는 여러 컨트롤러 노드에서 동일한 서비스를 실행하고 HAProxy를 사용하여 단일 IP 주소를 사용하여 노드 또는 특정 컨트롤러에 트래픽을 배포합니다. HAProxy는 경우에 따라 roundRbin 스케줄링을 사용하여 활성-활성 서비스에 트래픽을 분산합니다. 더 많은 컨트롤러 노드를 추가하여 성능을 향상시킬 수 있습니다.

- **active-passive**: active-active 모드에서 실행할 수 없는 서비스는 active-passive 모드로 실행되어야 합니다. 이 모드에서는 한 번에 하나의 서비스 인스턴스만 활성화됩니다. 예를 들어 HAProxy는 고정 테이블 옵션을 사용하여 들어오는 Galera 데이터베이스 연결 요청을 단일 백엔드 서비스로 전달합니다. 이렇게 하면 여러 Galera 노드에서 동일한 데이터에 너무 많은 동시 연결을 방지할 수 있습니다.

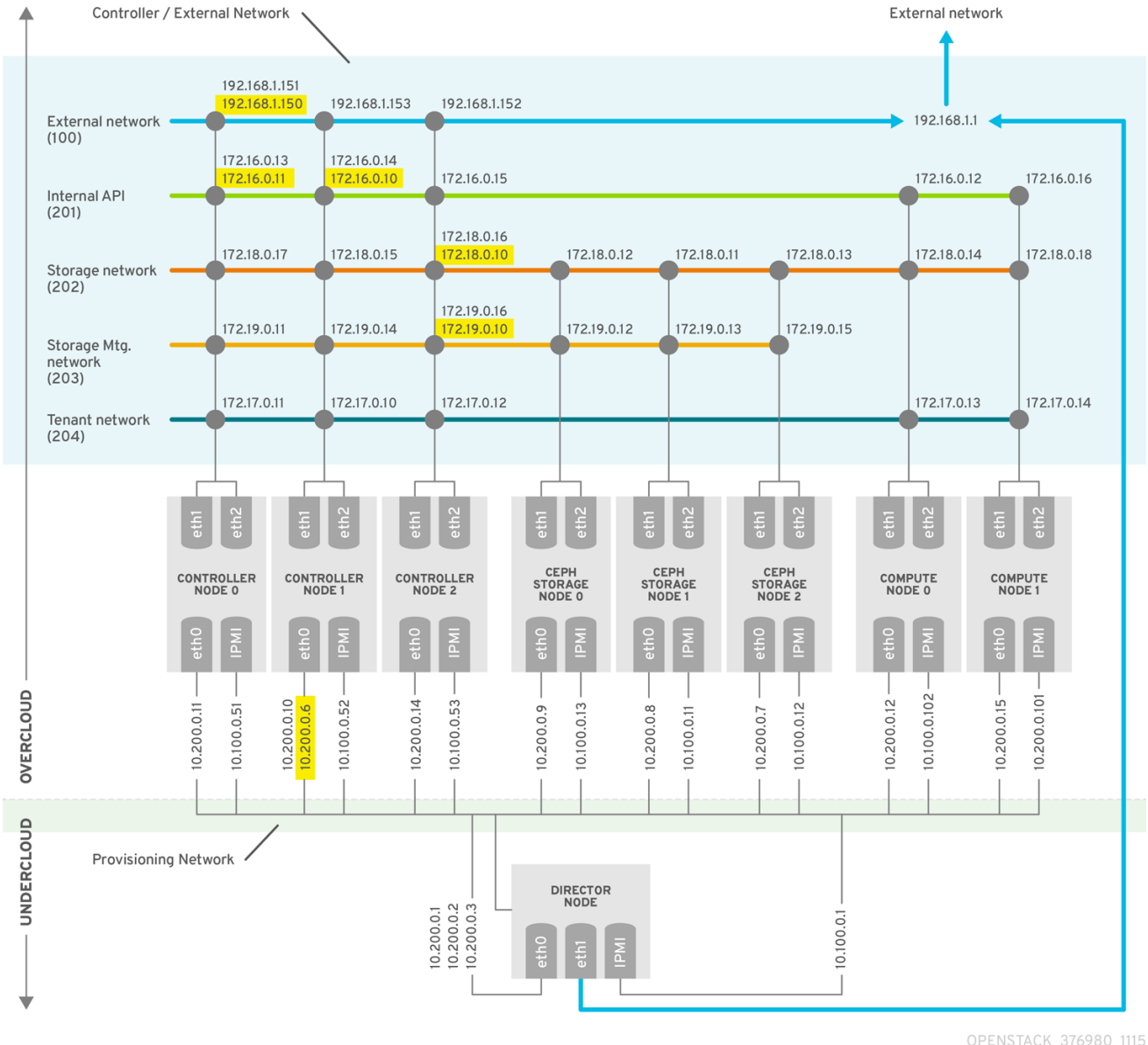
2장. 배포 예: COMPUTE 및 CEPH가 있는 고가용성 클러스터

다음 예제 시나리오에서는 OpenStack Compute 서비스 및 Red Hat Ceph Storage를 사용하는 고가용성 배포에 필요한 아키텍처, 하드웨어 및 네트워크 사양과 언더클라우드 및 오버클라우드 구성 파일을 보여줍니다.

중요

이 배포는 테스트 환경에 대한 참조로 사용하기 위한 것이며 프로덕션 환경에서 지원되지 않습니다.

그림 2.1. 고가용성 배포 아키텍처 예



Red Hat Ceph Storage 클러스터 배포에 대한 자세한 내용은 Deploying an Overcloud with Containerized Red Hat Ceph 를 참조하십시오.

director를 사용하여 Red Hat OpenStack Platform을 배포하는 방법에 대한 자세한 내용은 Director Installation and Usage 를 참조하십시오.

2.1. 하드웨어 사양

다음 표는 예제 배포에 사용된 하드웨어를 보여줍니다. 자체 테스트 배포에서 필요에 따라 CPU, 메모리, 스토리지 또는 NIC를 조정할 수 있습니다.

표 2.1. 물리적 컴퓨터

컴퓨터 수	목적	CPU	메모리	디스크 공간	전원 관리	nics
1	언더클라우드 노드	4	6144 MB	40GB	IPMI	2 (1 외부; 프로비저닝 시 1개 + 1 IPMI)
3	컨트롤러 노드	4	6144 MB	40GB	IPMI	3 (2개 오버클라우드에서 결합됨, 프로비저닝 시 1개) + 1 IPMI
3	Ceph Storage 노드	4	6144 MB	40GB	IPMI	3 (2개 오버클라우드에서 결합됨, 프로비저닝 시 1개) + 1 IPMI
2	컴퓨팅 노드 (필요한 추가)	4	6144 MB	40GB	IPMI	3 (2개 오버클라우드에서 결합됨, 프로비저닝 시 1개) + 1 IPMI

하드웨어 할당을 계획할 때 다음 지침을 검토하십시오.

컨트롤러 노드

대부분의 비스토리지 서비스는 컨트롤러 노드에서 실행됩니다. 모든 서비스는 세 개의 노드에 복제되며 활성-활성 또는 활성-수동 서비스로 구성됩니다. HA 환경에는 최소 3개의 노드가 필요합니다.

Red Hat Ceph Storage 노드

스토리지 서비스는 이러한 노드에서 실행되며 Red Hat Ceph Storage 영역 풀을 컴퓨팅 노드에 제공합니다. 최소 3개의 노드가 필요합니다.

컴퓨팅 노드

VM(가상 머신) 인스턴스는 컴퓨팅 노드에서 실행됩니다. 용량 요구 사항과 마이그레이션 및 재부팅 작업을 충족하기 위해 필요한 만큼 컴퓨팅 노드를 배포할 수 있습니다. VM이 스토리지 노드, 다른 컴퓨팅 노드의 VM 및 공용 네트워크에 액세스할 수 있도록 컴퓨팅 노드를 스토리지 네트워크 및 테넌트 네트워크에 연결해야 합니다.

STONITH

고가용성 오버클라우드에서 Pacemaker 클러스터의 일부인 각 노드에 대해 STONITH 장치를 설정해야 합니다. STONITH를 사용하지 않는 고가용성 오버클라우드 배포는 지원되지 않습니다. STONITH 및 Pacemaker에 관한 자세한 내용은 Fencing in a Red Hat High Availability Cluster 및 Support

Policies for RHEL High Availability Clusters를 참조하십시오.

2.2. 네트워크 사양

다음 표는 예제 배포에서 사용되는 네트워크 구성을 보여줍니다.

참고

이 예제에는 컨트롤 플레인의 하드웨어 중복과 overcloud keystone 관리 엔드포인트가 구성된 프로비저닝 네트워크는 포함되어 있지 않습니다.

표 2.2. 물리적 및 가상 네트워크

물리적 NIC	목적	VLAN	설명
eth0	프로비저닝 네트워크 (undercloud)	해당 없음	director(undercloud)에서 모든 노드 관리
eth1 및 eth2	controller/External(overcloud)	해당 없음	VLAN을 사용한 본딩 NIC
	외부 네트워크	VLAN 100	환경 외부에서 테넌트 네트워크, 내부 API 및 OpenStack Horizon 대시보드에 대한 액세스 허용
	내부 API	VLAN 201	컴퓨팅 노드와 컨트롤러 노드 간의 내부 API에 대한 액세스 제공
	스토리지 액세스	VLAN 202	컴퓨팅 노드를 스토리지 미디어에 연결
	스토리지 관리	VLAN 203	스토리지 미디어 관리
	테넌트 네트워크	VLAN 204	RHOSP에 테넌트 네트워크 서비스 제공

네트워크 구성 외에도 다음 구성 요소를 배포해야 합니다.

프로비저닝 네트워크 스위치

- 이 스위치는 언더클라우드를 오버클라우드의 모든 실제 컴퓨터에 연결할 수 있어야 합니다.
- 이 스위치에 연결된 각 오버클라우드 노드의 NIC는 언더클라우드에서 PXE 부팅할 수 있어야 합니다.
- **portfast** 매개변수를 활성화해야 합니다.

컨트롤러/외부 네트워크 스위치

- 이 스위치는 배포의 다른 VLAN에 대해 VLAN 태깅을 수행하도록 구성해야 합니다.
- VLAN 100개의 트래픽만 외부 네트워크로 허용합니다.

네트워킹 하드웨어 및 keystone 엔드포인트

- 오버클라우드 서비스의 가용성을 차단하는 컨트롤러 노드 네트워크 카드 또는 네트워크 스위치 실패를 방지하려면 결합된 네트워크 카드 또는 네트워킹 하드웨어 중복을 사용하는 네트워크에 keystone 관리 엔드포인트가 있는지 확인합니다.
  keystone 엔드포인트를 **internal_api** 등 다른 네트워크로 이동하는 경우, 언더클라우드가 VLAN 또는 서브넷에 연결할 수 있는지 확인합니다. 자세한 내용은 Red Hat Knowledgebase 솔루션에서 Keystone 관리 엔드포인트를 internal_api 네트워크로 마이그레이션하는 방법 을 참조하십시오.

2.3. 언더클라우드 설정 파일

예제 배포에서는 다음 언더클라우드 구성 파일을 사용합니다.

instackenv.json

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

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

```
        "cpu": "2",
        "pm_user": "admin"
       },
       {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.102",
        "mac": [
         "2c:c2:60:0d:e7:d1"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "2",
        "pm_user": "admin"
       }
     ],
     "overcloud": {"password": "7adbbbeedc5b7a07ba1917e1b3b228334f9a2d4e",
     "endpoint": "http://192.168.1.150:5000/v2.0/"
               }
}
```

undercloud.conf

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

network-environment.yaml

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

parameter_defaults:
```

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

2.4. 오버클라우드 설정 파일

예제 배포에서는 다음 오버클라우드 구성 파일을 사용합니다.

/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg (Controller 노드)

이 파일은 HAProxy가 관리하는 서비스를 식별합니다. 여기에는 HAProxy에서 모니터링하는 서비스의 설정이 포함되어 있습니다. 이 파일은 모든 컨트롤러 노드에서 동일합니다.

```
# This file is managed by Puppet
global
  daemon
  group  haproxy
  log  /dev/log local0
  maxconn  20480
  pidfile  /var/run/haproxy.pid
  ssl-default-bind-ciphers
!SSLv2:kEECDH:kRSA:kEDH:kPSK:+3DES:!aNULL:!eNULL:!MD5:!EXP:!RC4:!SEED:!IDEA:!DES
  ssl-default-bind-options  no-sslv3
  stats  socket /var/lib/haproxy/stats mode 600 level user
  stats  timeout 2m
  user  haproxy

defaults
  log  global
  maxconn  4096
  mode  tcp
  retries  3
  timeout  http-request 10s
  timeout  queue 2m
  timeout  connect 10s
```

```
  timeout  client 2m
  timeout  server 2m
  timeout  check 10s

listen aodh
  bind 192.168.1.150:8042 transparent
  bind 172.16.0.10:8042 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8042 check fall 5 inter 2000 rise 2

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

listen glance_api
  bind 192.168.1.150:9292 transparent
  bind 172.18.0.10:9292 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /healthcheck
  server overcloud-controller-0.internalapi.localdomain 172.18.0.17:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.18.0.15:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.18.0.16:9292 check fall 5 inter 2000 rise 2

listen gnocchi
  bind 192.168.1.150:8041 transparent
  bind 172.16.0.10:8041 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8041 check fall 5 inter 2000 rise 2

listen haproxy.stats
  bind 10.200.0.6:1993 transparent
  mode http
  stats enable
  stats uri /
  stats auth admin:PnDD32EzdVCf73CpjHhFGHZdV

listen heat_api
```

```
  bind 192.168.1.150:8004 transparent
  bind 172.16.0.10:8004 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  timeout client 10m
  timeout server 10m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8004 check fall 5 inter 2000 rise 2

listen heat_cfn
  bind 192.168.1.150:8000 transparent
  bind 172.16.0.10:8000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  timeout client 10m
  timeout server 10m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8000 check fall 5 inter 2000 rise 2

listen horizon
  bind 192.168.1.150:80 transparent
  bind 172.16.0.10:80 transparent
  mode http
  cookie SERVERID insert indirect nocache
  option forwardfor
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2

listen keystone_admin
  bind 192.168.24.15:35357 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /v3
  server overcloud-controller-0.ctlplane.localdomain 192.168.24.9:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.ctlplane.localdomain 192.168.24.8:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.ctlplane.localdomain 192.168.24.18:35357 check fall 5 inter 2000 rise
2

listen keystone_public
  bind 192.168.1.150:5000 transparent
  bind 172.16.0.10:5000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

```
  option httpchk GET /v3
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:5000 check fall 5 inter 2000 rise 2

listen mysql
  bind 172.16.0.10:3306 transparent
  option tcpka
  option httpchk
  stick on dst
  stick-table type ip size 1000
  timeout client 90m
  timeout server 90m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200

listen neutron
  bind 192.168.1.150:9696 transparent
  bind 172.16.0.10:9696 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:9696 check fall 5 inter 2000 rise 2

listen nova_metadata
  bind 172.16.0.10:8775 transparent
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8775 check fall 5 inter 2000 rise 2

listen nova_novncproxy
  bind 192.168.1.150:6080 transparent
  bind 172.16.0.10:6080 transparent
  balance source
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option tcpka
  timeout tunnel 1h
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6080 check fall 5 inter 2000 rise 2

listen nova_osapi
  bind 192.168.1.150:8774 transparent
  bind 172.16.0.10:8774 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

```
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8774 check fall 5 inter 2000 rise 2

listen nova_placement
  bind 192.168.1.150:8778 transparent
  bind 172.16.0.10:8778 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8778 check fall 5 inter 2000 rise 2

listen panko
  bind 192.168.1.150:8977 transparent
  bind 172.16.0.10:8977 transparent
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8977 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8977 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8977 check fall 5 inter 2000 rise 2

listen redis
  bind 172.16.0.13:6379 transparent
  balance first
  option tcp-check
  tcp-check send AUTH\ V2EgUh2pvkr8VzU6yuE4XHsr9\r\n
  tcp-check send PING\r\n
  tcp-check expect string +PONG
  tcp-check send info\ replication\r\n
  tcp-check expect string role:master
  tcp-check send QUIT\r\n
  tcp-check expect string +OK
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6379 check fall 5 inter 2000 rise 2

listen swift_proxy_server
  bind 192.168.1.150:8080 transparent
  bind 172.18.0.10:8080 transparent
  option httpchk GET /healthcheck
  timeout client 2m
  timeout server 2m
  server overcloud-controller-0.storage.localdomain 172.18.0.17:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.storage.localdomain 172.18.0.15:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.storage.localdomain 172.18.0.16:8080 check fall 5 inter 2000 rise 2
```

/etc/corosync/corosync.conf 파일 (Controller 노드)

이 파일은 클러스터 인프라를 정의하고 모든 컨트롤러 노드에서 사용할 수 있습니다.

```
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 노드)

이 파일에는 모니터링 호스트의 호스트 이름 및 IP 주소를 포함한 Ceph 고가용성 설정이 포함되어 있습니다.

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

3장. 고가용성 환경 액세스

언더클라우드에서 특정 HA 노드에 액세스하고 조사할 수 있습니다.

절차

1. 실행 중인 HA 환경에서 언더클라우드에 로그인합니다.
2. **stack** 사용자로 변경합니다.

```
# sudo su - stack
```

3. 언더클라우드 노드에 액세스하여 조사하려면 언더클라우드에서 노드의 IP 주소를 가져옵니다.

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

4. 오버클라우드 노드 중 하나에 로그인하려면 다음 명령을 실행합니다.

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

이름과 IP 주소를 배포의 실제 값으로 바꿉니다.

4장. PACEMAKER를 사용하여 고가용성 서비스 관리

Pacemaker 서비스는 Galera, RabbitMQ, Redis 및 HAProxy와 같은 핵심 컨테이너 및 활성-수동 서비스를 관리합니다. Pacemaker를 사용하여 관리 서비스, 가상 IP 주소, 전원 관리 및 펜싱에 대한 일반 정보를 보고 관리합니다.

Red Hat Enterprise Linux의 Pacemaker에 대한 자세한 내용은 Red Hat Enterprise Linux 설명서에서 고가용성 클러스터 구성 및 관리를 참조하십시오.

4.1. 리소스 번들 및 컨테이너

Pacemaker는 RHOSP(Red Hat OpenStack Platform) 서비스를 *Bundle Set 리소스* 또는 *번들로* 관리합니다. 이러한 서비스 대부분은 동일한 방식으로 시작되고 항상 각 컨트롤러 노드에서 실행되는 활성-활성 서비스입니다.

Pacemaker는 다음 리소스 유형을 관리합니다.

번들

번들 리소스는 모든 컨트롤러 노드에서 동일한 컨테이너를 구성하고 복제하고, 필요한 스토리지 경로를 컨테이너 디렉터리에 매핑하고, 리소스 자체와 관련된 특정 속성을 설정합니다.

컨테이너

컨테이너는 HAProxy와 같은 간단한 **systemd** 서비스에서 Galera와 같은 복잡한 서비스에 이르기까지 다양한 종류의 리소스를 실행할 수 있습니다. 이 서비스에는 서로 다른 노드에서 서비스 상태를 제어하고 설정하는 특정 리소스 에이전트가 필요합니다.

중요

- **docker** 또는 **systemctl** 을 사용하여 번들 또는 컨테이너를 관리할 수 없습니다. 명령을 사용하여 서비스 상태를 확인할 수 있지만 Pacemaker를 사용하여 이러한 서비스에서 작업을 수행해야 합니다.
- Pacemaker에서 제어하는 Docker 컨테이너에는 Docker에서 제공하는 **RestartPolicy** 가 **no** 로 설정되어 있습니다. 이는 Docker가 아닌 Pacemaker가 컨테이너 시작 및 중지 작업을 제어하도록 하기 위한 것입니다.

Simple Bundle Set 리소스(간단 번들)

간단한 Bundle Set 리소스 또는 *간단한 번들은* 각각 컨트롤러 노드에 배포하려는 동일한 Pacemaker 서비스를 포함하는 컨테이너 집합입니다.

다음 예제는 **pcs status** 명령의 출력에서 간단한 번들 목록을 보여줍니다.

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

각 번들에 대해 다음 세부 정보를 볼 수 있습니다.

- Pacemaker에서 서비스에 할당하는 이름입니다.
- 번들과 연결된 컨테이너에 대한 참조입니다.

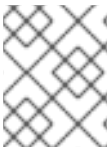
- 다른 컨트롤러 노드에서 실행 중인 복제본 목록 및 상태입니다.

다음 예제는 **haproxy-bundle** 간단한 번들의 설정을 보여줍니다.

```
$ sudo pcs resource show haproxy-bundle
Bundle: haproxy-bundle
 Docker: image=192.168.24.1:8787/rhosp-rhel7/openstack-haproxy:pcmklatest network=host
options="--user=root --log-driver=journald -e KOLLA_CONFIG_STRATEGY=COPY_ALWAYS"
replicas=3 run-command="/bin/bash /usr/local/bin/kolla_start"
 Storage Mapping:
  options=ro source-dir=/var/lib/kolla/config_files/haproxy.json target-
dir=/var/lib/kolla/config_files/config.json (haproxy-cfg-files)
  options=ro source-dir=/var/lib/config-data/puppet-generated/haproxy/ target-
dir=/var/lib/kolla/config_files/src (haproxy-cfg-data)
  options=ro source-dir=/etc/hosts target-dir=/etc/hosts (haproxy-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (haproxy-localtime)
  options=ro source-dir=/etc/pki/ca-trust/extracted target-dir=/etc/pki/ca-trust/extracted (haproxy-pki-
extracted)
  options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-dir=/etc/pki/tls/certs/ca-bundle.crt
(haproxy-pki-ca-bundle-crt)
  options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-dir=/etc/pki/tls/certs/ca-
bundle.trust.crt (haproxy-pki-ca-bundle-trust-crt)
  options=ro source-dir=/etc/pki/tls/cert.pem target-dir=/etc/pki/tls/cert.pem (haproxy-pki-cert)
  options=rw source-dir=/dev/log target-dir=/dev/log (haproxy-dev-log)
```

이 예제에서는 번들의 컨테이너에 대한 다음 정보를 보여줍니다.

- **Image**: 언더클라우드에서 로컬 레지스트리를 참조하는 컨테이너에서 사용하는 이미지입니다.
- **Network**: 예의 **"host"** 인 컨테이너 네트워크 유형입니다.
- **options**: 컨테이너에 대한 특정 옵션
- **replicas**: 클러스터에서 실행해야 하는 컨테이너 복사본 수를 나타냅니다. 각 번들에는 컨트롤러 노드당 하나씩 세 개의 컨테이너가 포함되어 있습니다.
- **run-command**: 컨테이너를 생성하는 데 사용되는 시스템 명령
- **storage Mapping**: 각 호스트의 로컬 경로를 컨테이너에 매핑합니다. 호스트에서 **haproxy** 구성을 확인하려면 **/etc/haproxy/haproxy.cfg** 파일 대신 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 파일을 엽니다.



참고

HAProxy는 선택한 서비스에 대한 부하 분산 트래픽을 통해 고가용성 서비스를 제공하지만 HAProxy를 Pacemaker 번들 서비스로 관리하여 고가용성 서비스로 구성합니다.

복잡한 Bundle Set 리소스(복합 번들)

복잡한 번들 설정 리소스 *또는 복잡한 번* 들은 간단한 번들에 포함된 기본 컨테이너 구성 외에도 리소스 구성을 지정하는 Pacemaker 서비스입니다.

이 구성은 실행되는 컨트롤러 노드에 따라 다른 상태를 가질 수 있는 서비스인 *Multi-State* 리소스를 관리하는 데 필요합니다.

이 예제에서는 **pcs status** 명령의 출력에서 복잡한 번들 목록을 보여줍니다.

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

이 출력은 각 복잡한 번들에 대한 다음 정보를 보여줍니다.

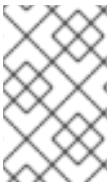
- RabbitMQ: 세 개의 컨트롤러 노드는 간단한 번들과 유사하게 서비스의 독립 실행형 인스턴스를 실행합니다.
- Galera: 세 개의 모든 컨트롤러 노드는 동일한 제약 조건에서 Galera 마스터로 실행됩니다.
- Redis: **overcloud-controller-0** 컨테이너가 마스터로 실행되고 다른 두 컨트롤러 노드는 슬레이브로 실행됩니다. 각 컨테이너 유형은 다른 제약 조건에서 실행될 수 있습니다.

다음 예제에서는 **galera-bundle** 복잡한 번들의 설정을 보여줍니다.

```
[...]
Bundle: galera-bundle
 Docker: image=192.168.24.1:8787/rhosp-rhel7/openstack-mariadb:pcmklatest masters=3
network=host options="--user=root --log-driver=journald -e
KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=3 run-command="/bin/bash
/usr/local/bin/kolla_start"
 Network: control-port=3123
 Storage Mapping:
  options=ro source-dir=/var/lib/kolla/config_files/mysql.json target-
dir=/var/lib/kolla/config_files/config.json (mysql-cfg-files)
  options=ro source-dir=/var/lib/config-data/puppet-generated/mysql/ target-
dir=/var/lib/kolla/config_files/src (mysql-cfg-data)
  options=ro source-dir=/etc/hosts target-dir=/etc/hosts (mysql-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (mysql-localtime)
  options=rw source-dir=/var/lib/mysql target-dir=/var/lib/mysql (mysql-lib)
  options=rw source-dir=/var/log/mariadb target-dir=/var/log/mariadb (mysql-log-mariadb)
  options=rw source-dir=/dev/log target-dir=/dev/log (mysql-dev-log)
 Resource: galera (class=ocf provider=heartbeat type=galera)
  Attributes: additional_parameters=--open-files-limit=16384 cluster_host_map=overcloud-controller-
0:overcloud-controller-0.internalapi.localdomain;overcloud-controller-1:overcloud-controller-
1.internalapi.localdomain;overcloud-controller-2:overcloud-controller-2.internalapi.localdomain
enable_creation=true wsrep_cluster_address=gcomm://overcloud-controller-
0.internalapi.localdomain,overcloud-controller-1.internalapi.localdomain,overcloud-controller-
2.internalapi.localdomain
  Meta Attrs: container-attribute-target=host master-max=3 ordered=true
  Operations: demote interval=0s timeout=120 (galera-demote-interval-0s)
              monitor interval=20 timeout=30 (galera-monitor-interval-20)
              monitor interval=10 role=Master timeout=30 (galera-monitor-interval-10)
              monitor interval=30 role=Slave timeout=30 (galera-monitor-interval-30)
              promote interval=0s on-fail=block timeout=300s (galera-promote-interval-0s)
```

```
            start interval=0s timeout=120 (galera-start-interval-0s)
            stop interval=0s timeout=120 (galera-stop-interval-0s)
[...]
```

이 출력에서는 간단한 번들과 달리 **galera-bundle** 리소스에는 다중 상태 리소스의 모든 측면을 결정하는 명시적 리소스 구성이 포함되어 있음을 보여줍니다.



참고

서비스는 동시에 여러 컨트롤러 노드에서 실행할 수 있지만 컨트롤러 노드 자체는 해당 서비스에 연결하는 데 필요한 IP 주소에서 수신 대기하지 않을 수 있습니다. 서비스의 IP 주소를 확인하는 방법에 대한 자세한 내용은 4.4절. "가상 IP 주소 보기" 을 참조하십시오.

4.2. 일반 PACEMAKER 정보 보기

일반 Pacemaker 정보를 보려면 **pcs status** 명령을 사용합니다.

절차

1. 모든 컨트롤러 노드에 **heat-admin** 사용자로 로그인합니다.

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

2. **pcs status** 명령을 실행합니다.

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

출력 예:

```
Cluster name: tripleo_cluster
Stack: corosync
Current DC: overcloud-controller-1 (version 1.1.16-12.el7_4.5-94ff4df) - partition with quorum

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

12 nodes configured
37 resources configured

Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-controller-2 ]
GuestOnline: [ galera-bundle-0@overcloud-controller-0 galera-bundle-1@overcloud-
controller-1 galera-bundle-2@overcloud-controller-2 rabbitmq-bundle-0@overcloud-
controller-0 rabbitmq-bundle-1@overcloud-controller-1 rabbitmq-bundle-2@overcloud-
controller-2 redis-bundle-0@overcloud-controller-0 redis-bundle-1@overcloud-controller-1
redis-bundle-2@overcloud-controller-2 ]

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

출력의 주요 섹션에서는 클러스터에 대한 다음 정보를 보여줍니다.

- **Cluster name**: 클러스터의 이름입니다.
- **[NUM] 노드 구성**: 클러스터에 대해 구성된 노드 수입니다.

- **[NUM] 리소스 구성**: 클러스터에 대해 구성된 리소스 수입니다.
- **온라인**: 현재 온라인 상태인 컨트롤러 노드의 이름입니다.
- **GuestOnline**: 현재 온라인 상태인 게스트 노드의 이름입니다. 각 게스트 노드는 복잡한 Bundle Set 리소스로 구성됩니다. 번들 세트에 대한 자세한 내용은 4.1절. "리소스 번들 및 컨테이너" 을 참조하십시오.

4.3. 번들 상태 보기

언더클라우드 노드에서 번들 상태를 확인하거나 컨트롤러 노드 중 하나에 로그인하여 번들 상태를 직접 확인할 수 있습니다.

언더클라우드 노드에서 번들 상태 확인

다음 명령을 실행합니다.

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

출력 예:

```
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
```

출력은 **haproxy** 프로세스가 컨테이너 내에서 실행 중임을 보여줍니다.

컨트롤러 노드에서 번들 상태 확인

컨트롤러 노드에 로그인하고 다음 명령을 실행합니다.

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

출력 예:

```
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
42454      17819   17774  0 06:08 ?        00:00:21 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
root      288508  237714  0 07:04 pts/0    00:00:00 grep --color=auto haproxy*
[root@controller-0 ~]# ps -ef | grep -e 17774 -e 17819
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
42454      17819   17774  0 06:08 ?        00:00:22 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
root      301950  237714  0 07:07 pts/0    00:00:00 grep --color=auto -e 17774 -e 17819
```

4.4. 가상 IP 주소 보기

각 IPaddr2 리소스는 클라이언트가 서비스에 대한 액세스를 요청하는 데 사용하는 가상 IP 주소를 설정합니다. 해당 IP 주소가 있는 컨트롤러 노드가 실패하면 IPaddr2 리소스에서 IP 주소를 다른 컨트롤러 노드에 다시 할당합니다.

모든 가상 IP 주소 표시

가상IP 유형을 사용하는 모든 리소스를 표시하려면 **--full** 옵션과 함께 **pcs resource show** 명령을 실행합니다.

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

다음 예제 출력에서는 현재 특정 가상 IP 주소를 수신하도록 설정된 각 컨트롤러 노드를 보여줍니다.

```
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
```

각 IP 주소는 처음에 특정 컨트롤러 노드에 연결됩니다. 예를 들어 **192.168.1.150** 은 **overcloud-controller-0** 에서 시작됩니다. 그러나 해당 컨트롤러 노드에 실패하면 IP 주소가 클러스터의 다른 컨트롤러 노드에 다시 할당됩니다.

다음 표에서는 예제 출력의 IP 주소를 설명하고 각 IP 주소의 원래 할당을 보여줍니다.

표 4.1. IP 주소 설명 및 할당 소스

IP 주소	설명	할당됨 from
192.168.1.150	공용 IP 주소	**network-environment.yaml** 파일의 **ExternalAllocationPools** 속성
10.200.0.6	컨트롤러 가상 IP 주소	**undercloud.conf** 파일에서 **dhcp_start** 및 **dhcp_end** 범위의 일부는 **10.200.0.5-10.200.0.24** 로 설정
172.16.0.10	컨트롤러 노드에서 OpenStack API 서비스에 대한 액세스 제공	**network-environment.yaml** 파일의 **InternalApiAllocationPools**
172.18.0.10	Glance API 및 Swift Proxy 서비스에 대한 액세스를 제공하는 스토리지 가상 IP 주소	**network-environment.yaml** 파일의 **StorageAllocationPools** 속성
172.16.0.11	컨트롤러 노드에서 Redis 서비스에 대한 액세스 제공	**network-environment.yaml** 파일의 **InternalApiAllocationPools**
172.19.0.10	스토리지 관리에 대한 액세스 제공	**network-environment.yaml** 파일의 **StorageMgmtAlloctionPools**

특정 IP 주소 보기

pcs resource show 명령을 실행합니다.

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

출력 예:

```
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)
```

특정 IP 주소에 대한 네트워크 정보 보기

1. 보려는 IP 주소에 할당된 컨트롤러 노드에 로그인합니다.
2. **ip addr show** 명령을 실행하여 네트워크 인터페이스 정보를 확인합니다.

```
$ ip addr show vlan100
```

출력 예:

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

3. **netstat** 명령을 실행하여 IP 주소를 청취하는 모든 프로세스를 표시합니다.

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

출력 예:

```
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
```

참고

0.0.0.0 과 같은 모든 로컬 주소를 청취하는 프로세스는 **192.168.1.150** 에서도 사용할 수 있습니다. 이러한 프로세스에는 **sshd**,**mysqld**,**dhclient**, pxe 가**포함됩니다**.

포트 번호 할당 보기

/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg 파일을 열어 기본 포트 번호 할당을 확인합니다.

다음 예제에서는 포트 번호와 수신 대기하는 서비스를 보여줍니다.

- TCP 포트 6080: **nova_novncproxy**
- TCP 포트 9696: **neutron**
- TCP 포트 8000: **heat_cfn**
- TCP 포트 80: **지평**
- TCP 포트 8776: **cinder**

이 예에서 **haproxy.cfg** 파일에 정의된 대부분의 서비스는 세 개의 컨트롤러 노드에서 **192.168.1.150** IP 주소를 수신 대기합니다. 그러나 **controller-0** 노드만 **192.168.1.150** IP 주소를 외부에서 수신 대기하고 있습니다.

따라서 **controller-0** 노드가 실패하는 경우 HAProxy는 다른 컨트롤러 노드에 **192.168.1.150** 만 다시 할당하고 다른 모든 서비스는 대체 컨트롤러 노드에서 이미 실행되고 있습니다.

4.5. PACEMAKER 상태 및 전원 관리 정보 보기

pcs status 출력의 마지막 섹션에서는 IPMI와 같은 전원 관리 펜싱에 대한 정보와 Pacemaker 서비스 자체의 상태를 표시합니다.

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

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

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

  pcsd: active/enabled
```

my-ipmilan-for-controller 설정은 각 컨트롤러 노드의 펜싱 유형(**stonith:fence_ipmilan**)과 IPMI 서비스가 중지되었는지 여부를 표시합니다. PCSD 상태는 세 개의 컨트롤러 노드가 모두 현재 온라인 상태임을 보여줍니다. Pacemaker 서비스는 **corosync**,**pacemaker**, **pcsd** 의 세 가지 데몬으로 구성됩니다. 이 예제에서는 세 개의 서비스가 모두 활성이며 활성화됩니다.

4.6. 실패한 PACEMAKER 리소스 문제 해결

Pacemaker 리소스가 실패하면 **pcs status** 출력의 **Failed Actions** 섹션을 볼 수 있습니다. 다음 예에서 **openstack-cinder-volume** 서비스는 **controller-0** 에서 작동을 중지했습니다.

```
Failed Actions:
* openstack-cinder-volume_monitor_60000 on overcloud-controller-0 'not running' (7): call=74,
```

```
status=complete, exitreason='none',
 last-rc-change='Wed Dec 14 08:33:14 2016', queued=0ms, exec=0ms
```

이 경우 systemd 서비스 *openstack-cinder-volume* 을 활성화해야 합니다. 다른 경우에는 문제를 찾아서 수정한 다음 리소스를 정리해야 할 수 있습니다. 리소스 문제 해결에 대한 자세한 내용은 8장. *리소스 문제 해결* 을 참조하십시오.

5장. STONITH를 사용하여 컨트롤러 노드 펜싱

펜싱은 클러스터 및 클러스터 리소스를 보호하기 위해 실패한 노드를 격리하는 프로세스입니다. 펜싱이 없으면 장애가 발생한 노드가 클러스터에 데이터 손상이 발생할 수 있습니다.

director는 Pacemaker를 사용하여 고가용성 컨트롤러 노드 클러스터를 제공합니다. Pacemaker는 *STONITH* 라는 프로세스를 사용하여 실패한 노드를 펜싱합니다. STONITH는 "스위에 있는 다른 노드 덕분"에 대한 약자입니다.

컨트롤러 노드가 상태 확인에 실패하면 Pacemaker 지정 조정자(DC) 역할을 하는 컨트롤러 노드는 Pacemaker **stonith** 서비스를 사용하여 영향을 받는 컨트롤러 노드를 펜싱합니다.

STONITH는 기본적으로 비활성화되어 있으며 Pacemaker에서 클러스터의 각 노드의 전원 관리를 제어할 수 있도록 수동 구성이 필요합니다.

중요

STONITH를 사용하지 않는 고가용성 오버클라우드 배포는 지원되지 않습니다. 고가용성 오버클라우드에서 Pacemaker 클러스터의 일부인 각 노드에 대해 STONITH 장치를 설정해야 합니다. STONITH 및 Pacemaker에 관한 자세한 내용은 Fencing in a Red Hat High Availability Cluster 및 Support Policies for RHEL High Availability Clusters 를 참조하십시오.

Red Hat Enterprise Linux에서 Pacemaker를 사용한 펜싱 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 고가용성 애드온 개요
- 고가용성 애드온 관리
- 고가용성 애드온 참조

5.1. 지원되는 펜싱 에이전트

펜싱을 사용하여 고가용성 환경을 배포할 때 환경 요구 사항에 따라 다음 펜싱 에이전트 중 하나를 선택할 수 있습니다. 펜싱 에이전트를 변경하려면 5.2절. “오버클라우드에서 펜싱 배포 및 테스트” 에 설명된 대로 **fencing.yaml** 파일에 추가 매개변수를 구성해야 합니다.

IPMI(Intelligent Platform Management Interface)

RHOSP에서 펜싱을 관리하는 데 사용하는 기본 펜싱 메커니즘

스토리지 블록 장치(SBD)

Watchdog 장치와 함께 배포에서 사용합니다. 배포에서는 공유 스토리지를 사용하지 않아야 합니다.

fence_kdump

kdump 충돌 복구 서비스와 함께 배포에서 를 사용합니다. 이 에이전트를 선택하는 경우 덤프 파일을 저장할 충분한 디스크 공간이 있는지 확인하십시오.
IPMI, **fence_rhevm** 또는 Redfish 펜싱 에이전트 외에도 이 에이전트를 보조 메커니즘으로 구성할 수 있습니다. 여러 펜싱 에이전트를 구성하는 경우 다음 작업을 시작하기 전에 첫 번째 에이전트가 작업을 완료할 수 있는 충분한 시간을 할당해야 합니다.

Redfish

DMTF Redfish API를 지원하는 서버와 함께 배포 시 사용합니다. 이 에이전트를 지정하려면 **fencing.yaml** 파일에서 **agent** 매개 변수의 값을 **fence_redfish** 로 변경합니다. Redfish에 대한 자세한 내용은 DTMF 문서 를 참조하십시오.

RHV(Red Hat Virtualization)용 fence_rhevm

을 사용하여 RHV 환경을 실행하는 컨트롤러 노드의 펜싱을 구성합니다. IPMI 펜싱과 동일한 방식으로 **fencing.yaml** 파일을 생성할 수 있지만 RHV를 사용하려면 **nodes.json** 파일에 **pm_type** 매개변수를 정의해야 합니다.
기본적으로 **ssl_insecure** 매개변수는 자체 서명된 인증서를 수락하도록 설정됩니다. 보안 요구 사항에 따라 매개변수 값을 변경할 수 있습니다.

중요

UserVMManager 와 같은 RHV에서 가상 시스템을 생성하고 시작하기 위해 권한이 있는 역할을 사용해야 합니다.

5.2. 오버클라우드에서 펜싱 배포 및 테스트

펜싱 구성 프로세스에는 다음 단계가 포함됩니다.

1. STONITH 및 Pacemaker 상태 검토.
2. **fencing.yaml** 파일을 생성합니다.
3. 오버클라우드를 재배포하고 설정을 테스트합니다.

사전 요구 사항

director에서 컨트롤러 노드를 등록할 때 생성한 **nodes.json** 파일에 액세스할 수 있는지 확인합니다. 이 파일은 배포 중에 생성하는 **fencing.yaml** 파일에 필요한 입력입니다.

STONITH 및 Pacemaker 상태 검토

1. 각 컨트롤러 노드에 **heat-admin** 사용자로 로그인합니다.
2. 클러스터가 실행 중인지 확인합니다.

```
$ sudo pcs status
```

출력 예:

```
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. STONITH가 비활성화되어 있는지 확인합니다.

```
$ sudo pcs property show
```

출력 예:

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

fencing.yaml 환경 파일을 생성합니다.

다음 옵션 중 하나를 선택합니다.

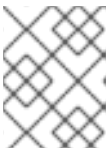
- IPMI 또는 RHV(Red Hat Virtualization) 펜싱 에이전트를 사용하는 경우 다음 명령을 실행하여 **fencing.yaml** 환경 파일을 생성합니다.

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

참고

- 이 명령은 **ilo** 및 **drac** 전원 관리 세부 정보를 IPMI로 변환합니다.
- **nodes.json** 파일에 노드의 NIC(네트워크 인터페이스) 중 하나의 MAC 주소가 포함되어 있는지 확인합니다. 자세한 내용은 오버클라우드 노드 등록을 참조하십시오.
- RHV를 사용하는 경우 권한이 있는 역할을 사용하여 **UserVMManager** 와 같은 가상 시스템을 생성하고 시작합니다.

- 스토리지 블록 장치(SBD), **fence_kdump** 또는 Redfish와 같은 다른 펜싱 에이전트를 사용하는 경우 **fencing.yaml** 파일을 수동으로 생성합니다.



참고

사전 프로비저닝된 노드를 사용하는 경우 **fencing.yaml** 파일도 수동으로 생성해야 합니다.

지원되는 펜싱 에이전트에 대한 자세한 내용은 5.1절. "지원되는 펜싱 에이전트" 을 참조하십시오.

오버클라우드를 재배포하고 설정을 테스트합니다.

1. **오버클라우드 배포** 명령을 실행하고 컨트롤러 노드에서 펜싱을 구성하기 위해 생성한 **fencing.yaml** 파일을 포함합니다.

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

2. 오버클라우드에 로그인하고 각 컨트롤러 노드에 펜싱이 구성되었는지 확인합니다.

a. Pacemaker가 리소스 관리자로 구성되어 있는지 확인합니다.

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

이 예에서 Pacemaker는 **fencing.yaml** 파일에 지정된 각 컨트롤러 노드에 STONITH 리소스를 사용하도록 구성되어 있습니다.

참고

제어하는 동일한 노드에서 **fence-resource** 프로세스를 구성해서는 안 됩니다.

b. **pcs stonith show** 명령을 실행하여 펜싱 리소스 특성을 확인합니다.

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

STONITH 특성 값은 **fencing.yaml** 파일의 값과 일치해야 합니다.

컨트롤러 노드에서 펜싱 확인

펜싱이 올바르게 작동하는지 테스트하려면 컨트롤러 노드의 모든 포트를 닫고 서버를 재부팅하여 펜싱을 트리거합니다.

1. 컨트롤러 노드에 로그인합니다.

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

2. **root** 사용자로 변경하고 각 포트에서 **iptables** 명령을 실행합니다.

```
$ 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
```

중요

이 단계에서는 컨트롤러 노드에 대한 모든 연결이 삭제되어 서버가 재부팅됩니다.

3. 다른 컨트롤러 노드에서 Pacemaker 로그 파일에서 펜싱 이벤트를 찾습니다.

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

- STONITH 서비스가 컨트롤러에서 펜싱 작업을 수행한 경우 로그 파일에 펜싱 이벤트가 표시됩니다.

4. 몇 분 정도 기다린 다음 **pcs status** 명령을 실행하여 재부팅된 컨트롤러 노드가 클러스터에서 다시 실행되고 있는지 확인합니다.

5.3. STONITH 정보 보기

STONITH가 펜싱 장치를 구성하는 방법을 보려면 오버클라우드에서 **pcs stonith show --full** 명령을 실행합니다.

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

full 옵션은 세 개의 컨트롤러 노드에 대한 펜싱 세부 정보를 반환합니다.

이 출력에서는 각 리소스에 대한 다음 정보를 보여줍니다.

- 펜싱 장치에서 필요에 따라 시스템을 켜거나 켜는 IPMI 전원 관리 서비스(예: **fence_ipmilan**)입니다.
- IPMI 인터페이스의 IP 주소(예: 10. **100.0.51)입니다**.
- **admin** 과 같이 로그인할 사용자 이름입니다.
- 노드에 로그인하는 데 사용할 암호(예: **abc**)입니다.
- 각 호스트가 **60s** 와 같이 모니터링되는 간격(초)입니다.

5.4. 펜싱 매개변수

오버클라우드에 펜싱을 배포할 때 펜싱을 구성하는 데 필요한 매개변수를 사용하여 **fencing.yaml** 파일을 생성합니다. 펜싱 배포 및 테스트에 대한 자세한 내용은 5.2절. "오버클라우드에서 펜싱 배포 및 테스트" 을 참조하십시오.

다음 예제는 **fencing.yaml** 환경 파일의 구조를 보여줍니다.

```
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
```

이 파일에는 다음 매개변수가 포함되어 있습니다.

EnableFencing

Pacemaker 관리 노드의 펜싱 기능을 활성화합니다.

FencingConfig

각 장치의 펜싱 장치와 매개 변수를 나열합니다.

- **agent**: 에이전트 이름 펜싱. Red Hat OpenStack Platform은 IPMI에 대해 **fence_ipmilan** 만 지원합니다.
- **host_mac**: 프로비저닝 인터페이스 또는 서버의 다른 네트워크 인터페이스의 소문자로 있는 mac 주소입니다. 이 ID를 펜싱 장치의 고유 식별자로 사용할 수 있습니다.
- **params**: 펜싱 장치 매개변수 목록입니다.

펜싱 장치 매개변수

- **auth**: IPMI 인증 유형(**md5**,**암호** 또는 none).
- **ipaddr**: IPMI IP 주소
- **ipport**: IPMI 포트.
- **login**: IPMI 장치의 사용자 이름입니다.
- **passwd**: IPMI 장치의 암호입니다.
- **lanplus**: 연결 보안을 개선하기 위해 lanplus를 사용합니다.
- **privlvl**: IPMI 장치의 권한 수준
- **mkmk_host_list**: Pacemaker 호스트 목록.

6장. HAPROXY를 사용하여 트래픽 부하 분산

HAProxy 서비스는 고가용성 클러스터의 컨트롤러 노드에 대한 트래픽 부하 분산과 로깅 및 샘플 구성을 제공합니다.

haproxy 패키지에는 동일한 이름의 **systemd** 서비스에 해당하는 **haproxy** 데몬이 포함되어 있습니다. Pacemaker는 HAProxy 서비스를 **haproxy-bundle** 이라는 고가용성 서비스로 관리합니다.

HAProxy에 대한 자세한 내용은 로드 밸런서 관리 가이드의 HAProxy 구성 장을 참조하십시오.

HAProxy가 올바르게 구성되었는지 확인하는 방법에 대한 자세한 내용은 KCS 문서 How can I verify my haproxy.cfg is correctly configured to load openstack services?.

6.1. HAPROXY 작동 방식

director는 HAProxy 서비스를 사용하도록 대부분의 Red Hat OpenStack Platform 서비스를 구성할 수 있습니다. director는 **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg** 파일에서 이러한 서비스를 설정하여 HAProxy가 각 오버클라우드 노드의 전용 컨테이너에서 실행되도록 지시합니다.

다음 표는 HAProxy에서 관리하는 서비스 목록을 보여줍니다.

표 6.1. HAProxy에서 관리하는 서비스

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

haproxy.cfg 파일의 각 서비스에 대해 다음 속성을 볼 수 있습니다.

- **listen**: 요청을 수신 대기 중인 서비스의 이름입니다.
- **bind**: 서비스가 수신 대기 중인 IP 주소 및 TCP 포트 번호입니다.
- **server**: HAProxy, IP 주소 및 수신 대기 포트, 서버에 대한 추가 정보를 사용하는 각 컨트롤러 노드 서버의 이름입니다.

다음 예제는 **haproxy.cfg** 파일의 OpenStack Block Storage(cinder) 서비스 구성을 보여줍니다.

```
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
```

이 예제 출력에서는 OpenStack Block Storage(cinder) 서비스에 대한 다음 정보를 보여줍니다.

- **172.16.0.10:8776**: 오버클라우드에서 사용할 내부 API 네트워크(VLAN201)의 가상 IP 주소 및 포트입니다.
- **192.168.1.150:8776**: 외부 네트워크(VLAN100)의 가상 IP 주소 및 포트로, 오버클라우드 외부에서 API 네트워크에 액세스할 수 있습니다.
- **8776**: OpenStack Block Storage(cinder) 서비스가 수신 대기 중인 포트 번호입니다.
- **Server**: 컨트롤러 노드 이름 및 IP 주소입니다. HAProxy는 해당 IP 주소로 이루어진 요청을 **서버** 출력에 나열된 컨트롤러 노드 중 하나로 보낼 수 있습니다.
- **httpchk**: 컨트롤러 노드 서버에서 상태 점검을 활성화합니다.
- **fall 5**: 서비스가 오프라인 상태인지 확인하는 데 실패한 상태 점검 수입니다.
- **2000년 간**: 연속된 두 개의 상태 점검 간격(밀리초)입니다.
- **up 2**: 서비스가 실행 중인지 확인하는 데 성공한 상태 점검 수입니다.

haproxy.cfg 파일에서 사용할 수 있는 설정에 대한 자세한 내용은 **haproxy** 패키지가 설치된 모든 노드의 **/usr/share/doc/haproxy-[VERSION]/configuration.txt** 파일을 참조하십시오.

6.2. HAPROXY 통계 보기

기본적으로 director는 모든 HA 배포에서 HAProxyions 또는 통계도 활성화합니다. 이 기능을 사용하면 HAProxy 순위 페이지의 데이터 전송, 연결 및 서버 상태에 대한 자세한 정보를 볼 수 있습니다.

director는 HAProxy statistics 페이지에 연결하고 **haproxy.cfg** 파일에 정보를 저장하는 데 사용하는 **IP:Port** 주소도 설정합니다.

절차

1. HAProxy가 설치된 모든 컨트롤러 노드에서 **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg** 파일을 엽니다.
2. **listen haproxy.stats** 섹션을 찾습니다.

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

3. 웹 브라우저에서 **10.200.0.6:1993** 으로 이동하여 **stats 인증** 행에서 자격 증명을 입력하여 HAProxy criterias 페이지를 확인합니다.

7장. GALERA를 사용하여 데이터베이스 복제 관리

Red Hat OpenStack Platform은 MariaDB Galera Cluster 를 사용하여 데이터베이스 복제를 관리합니다. Pacemaker는 데이터베이스 마스터/슬레이브 상태를 관리하는 **번들 세트** 리소스로 Galera 서비스를 실행합니다. Galera를 사용하여 호스트 이름 확인, 클러스터 무결성, 노드 무결성, 데이터베이스 복제 성능과 같은 데이터베이스 클러스터의 다양한 측면을 테스트하고 확인할 수 있습니다.

다른 Pacemaker 서비스와 유사하게 **pcs status** 명령을 사용하여 Galera 서비스가 실행 중인지 및 실행 중인 컨트롤러 노드를 확인할 수 있습니다. Pacemaker 번들 상태 보기에 대한 자세한 내용은 4.3절. "번들 상태 보기" 을 참조하십시오.

데이터베이스 클러스터 무결성을 조사할 때 각 노드는 다음 기준을 충족해야 합니다.

- 노드는 올바른 클러스터의 일부입니다.
- 노드는 클러스터에 쓸 수 있습니다.
- 노드는 클러스터에서 쿼리 및 쓰기 명령을 수신할 수 있습니다.
- 노드는 클러스터의 다른 노드에 연결되어 있습니다.
- 노드는 로컬 데이터베이스의 테이블에 쓰기 세트를 복제하고 있습니다.

7.1. 호스트 이름 확인

기본적으로 director는 Galera 리소스를 IP 주소 대신 호스트 이름에 바인딩합니다. 따라서 DNS를 잘못 구성하거나 실패한 DNS와 같이 호스트 이름 확인을 방지하는 문제로 인해 Pacemaker에서 Galera 리소스를 잘못 관리할 수 있습니다.

MariaDB Galera 클러스터의 문제를 해결하려면 먼저 호스트 이름 확인 문제를 제거한 다음 각 컨트롤러 노드의 데이터베이스에서 쓰기 세트 복제 상태를 확인합니다. MySQL 데이터베이스에 액세스하려면 오버클라우드 배포 중에 director에서 설정한 암호를 사용합니다.

절차

1. 컨트롤러 노드에서 **hiera** 명령을 실행하여 MariaDB 데이터베이스 루트 암호를 가져옵니다.

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

2. 노드에서 실행되는 MariaDB 컨테이너의 이름을 가져옵니다.

```
$ sudo docker ps | grep -i galera
5fb195b0d9e8        192.168.24.1:8787/rh-osbs/rhosp13-openstack-mariadb:pcmklatest
"dumb-init -- /bin..."   7 hours ago         Up 7 hours  galera-bundle-docker-0
```

3. 각 노드의 MariaDB 데이터베이스에서 쓰기 집합 복제 정보를 가져옵니다.

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

```
| wsrep_apply_oool          | 0.000630 |
| wsrep_apply_window        | 1.021942 |
| ...                       | ...      |
+---------------------------+----------+
```

각 관련 변수는 접두사 **wsrep** 를 사용합니다.

4. 먼저 클러스터가 올바른 노드 수를 보고하고 있는지 확인하여 MariaDB Galera 클러스터의 상태 및 무결성을 확인합니다.

7.2. 데이터베이스 클러스터 무결성 확인

MariaDB Galera Cluster에서 문제를 조사할 때 각 컨트롤러 노드에서 특정 **wsrep** 데이터베이스 변수를 확인하여 전체 클러스터의 무결성을 확인할 수 있습니다.

절차

다음 명령을 실행하고 확인할 **wsrep** 데이터베이스 변수로 VArichBLE을 바꿉니다.

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

*다음 예제는 노드의 클러스터 상태 **UUID**를 확인하는 방법을 보여줍니다.*

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

+--------------------------+--------------------------------------+
| Variable_name            | Value                                |
+--------------------------+--------------------------------------+
| wsrep_cluster_state_uuid | e2c9a15e-5485-11e0-0800-6bbb637e7211 |
+--------------------------+--------------------------------------+
```

*다음 표에는 클러스터 무결성을 확인하는 데 사용할 수 있는 **wsrep** 데이터베이스 변수가 나열되어 있습니다.*

*표 **7.1.** 클러스터 무결성을 확인하는 데이터베이스 변수*

변수	요약	설명
wsrep_cluster_state_uuid	클러스터 상태 UUID	노드가 속한 클러스터의 ID입니다. 모든 노드에는 동일한 클러스터 ID가 있어야 합니다. 다른 ID를 가진 노드는 클러스터에 연결되어 있지 않습니다.

변수	요약	설명
wsrep_cluster_size	클러스터의 노드 수	모든 노드에서 확인할 수 있습니다. 값이 실제 노드 수보다 작으면 일부 노드가 실패하거나 연결이 끊어진 것입니다.
wsrep_cluster_conf_id	총 클러스터 변경 수	클러스터가 여러 구성 요소(*파티션* 이라고도 함)로 분할되었는지 여부를 확인합니다. 파티션은 일반적으로 네트워크 오류로 인해 발생합니다. 모든 노드는 동일한 값을 가져야 합니다. 일부 노드가 다른 **wsrep_cluster_conf_id** 를 보고하는 경우 **wsrep_cluster_status** 값을 확인하여 노드가 여전히 클러스터에 쓸 수 있는지 확인합니다(**기본**).
wsrep_cluster_status	기본 구성 요소 상태	노드가 클러스터에 쓸 수 있는지 여부를 확인합니다. 노드가 클러스터에 쓸 수 있는 경우 **wsrep_cluster_status** 값은 **Primary** 입니다. 다른 값은 노드가 작동하지 않는 파티션의 일부임을 나타냅니다.

7.3. 데이터베이스 노드 무결성 확인

Galera 클러스터 문제를 특정 노드에 분리할 수 있는 경우 특정 wsrep 데이터베이스 변수는 노드의 특정 문제를 나타낼 수 있습니다.

절차

다음 명령을 실행하고 확인할 wsrep 데이터베이스 변수로 VArichBLE을 바꿉니다.

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

다음 표에는 노드의 무결성을 확인하는 데 사용할 수 있는 wsrep 데이터베이스 변수가 나열되어 있습니다.

표 7.2. 노드 무결성을 확인하는 데이터베이스 변수

변수	요약	설명
wsrep_ready	쿼리 허용 노드 기능	클러스터에서 노드의 쓰기 집합을 허용할 수 있는지의 여부입니다. 이 경우 **wsrep_ready** 가 **ON** 입니다.
wsrep_connected	노드 네트워크 연결	노드가 네트워크의 다른 노드에 연결할 수 있는지 여부를 나타냅니다. 이 경우 **wsrep_connected** 가 켜져 **있습니다**.
wsrep_local_state_comment	노드 상태	노드 상태가 요약됩니다. 노드가 클러스터에 쓸 수 있는 경우 **wsrep_local_state_comment** 에 대한 일반적인 값이 참여하거나 **SST** 대기, **조인**, 동기화된 또는 도고일 수 **있습니다**. 노드가 작동하지 않는 구성 요소의 일부인 경우 **wsrep_local_state_comment** 값은 **Initialized** 입니다.

참고

- *노드가 클러스터의 하위 집합에만 연결되어 있어도 **wsrep_connected** 값은 **ON** 일 수 있습니다. 예를 들어 클러스터 파티션의 경우 노드가 클러스터에 쓸 수 없는 구성 요소의 일부일 수 있습니다. 클러스터 무결성을 확인하는 방법에 대한 자세한 내용은 7.2절. "데이터베이스 클러스터 무결성 확인" 을 참조하십시오.*

- *wsrep_connected 값이 OFF 이면 노드가 클러스터 구성 요소에 연결되지 않습니다.*

7.4. 데이터베이스 복제 성능 테스트

클러스터와 개별 노드가 모두 정상이고 안정적인 경우 특정 데이터베이스 변수를 쿼리하여 복제 처리량에서 성능 벤치마크 테스트를 실행할 수 있습니다.

*이러한 변수 중 하나를 쿼리할 때마다 **FLUSH STATUS** 명령이 변수 값을 재설정합니다. 벤치마크 테스트를 실행하려면 여러 쿼리를 실행하고 분산을 분석해야 합니다. 이러한 변동은 클러스터의 성능에 영향을 미치는 흐름 제어 의 양을 결정하는 데 도움이 될 수 있습니다.*

*흐름 제어는 클러스터가 복제를 관리하는 데 사용하는 메커니즘입니다. 로컬 수신된 큐 가 특정 임계값을 초과하면 **Flow Control**은 대기열 크기가 중단될 때까지 복제를 일시 중지합니다. **Flow Control**에 대한 자세한 내용은 Galera Cluster 웹 사이트에서 Flow Control 을 참조하십시오.*

절차

*다음 명령을 실행하고 확인할 **wsrep** 데이터베이스 변수로 **VArichBLE**을 바꿉니다.*

```
$ sudo docker exec galera-bundle-docker-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW STATUS LIKE 'VARIABLE';"
```

*다음 표에는 데이터베이스 복제 성능을 테스트하는 데 사용할 수 있는 **wsrep** 데이터베이스 변수가 나열되어 있습니다.*

*표 **7.3.** 데이터베이스 복제 성능을 확인하는 데이터베이스 변수*

변수	요약	사용법
wsrep_local_recv_queue_avg	마지막 쿼리 이후의 로컬 수신된 쓰기 세트 대기열의 평균 크기입니다.	값이 0.0 보다 높으면 노드가 쓰기 세트를 수신할 때마다 write-sets를 적용할 수 없음을 나타냅니다. 이 경우 복제 제한을 트리거합니다. 이 벤치마크를 자세히 살펴보려면 **wsrep_local_recv_queue_min** 및 **wsrep_local_recv_queue_max** 를 확인합니다.
wsrep_local_send_queue_avg	마지막 쿼리 후 평균 대기열 길이입니다.	값이 0.0 보다 큰 경우 복제 제한 및 네트워크 처리량 문제가 발생할 가능성이 높습니다.
wsrep_local_recv_queue_min 및 **wsrep_local_recv_queue_max**	마지막 쿼리 후 로컬 수신 대기열의 최소 및 최대 크기입니다.	**wsrep_local_recv_queue_avg** 의 값이 0.0 보다 큰 경우 이러한 변수를 확인하여 큐 크기의 범위를 확인할 수 있습니다.

변수	요약	사용법
wsrep_flow_control_paused	마지막 쿼리 후 Flow Control이 노드를 일시 중지한 시간입니다.	값이 0.0 보다 큰 경우 Flow Control이 노드를 일시 중지했음을 나타냅니다. 일시 중지 기간을 결정하려면 **wsrep_flow_control_paused** 값을 쿼리 간에 초 단위로 곱합니다. 최적의 값은 0.0 에 최대한 가깝습니다. 예를 들어 다음과 같습니다. ● **wsrep_flow_control_paused** 값이 마지막 쿼리 후 0.50 1분이면 Flow Control이 노드를 30초 동안 일시 중지했습니다. ● **wsrep_flow_control_paused** 값이 마지막 쿼리 후 1.0 1분이면 Flow Control이 노드 전체 분 동안 일시 중지되었습니다.
wsrep_cert_deps_distance	병렬로 적용할 수 있는 가장 낮은 시퀀스 번호 (**seqno**) 값 간의 평균 차이점	제한 및 일시 중지의 경우 이 변수는 평균적으로 동시에 적용할 수 있는 쓰기 집합의 수를 나타냅니다. 값을 **wsrep_slave_threads** 변수와 비교하여 실제로 동시에 적용할 수 있는 쓰기 세트 수를 확인합니다.
wsrep_slave_threads	동시에 적용할 수 있는 스레드 수	이 변수의 값을 늘려 더 많은 스레드를 동시에 적용할 수 있으므로 **wsrep_cert_deps_distance** 값이 증가합니다. **wsrep_slave_threads** 값은 노드의 CPU 코어 수보다 클 수 없습니다. 예를 들어 **wsrep_cert_deps_distance** 값이 **20** 인 경우 **wsrep_slave_threads** 값을 **2** 에서 **4** 로 늘려 노드가 적용할 수 있는 쓰기 세트 양을 늘릴 수 있습니다. 문제가 있는 노드에 이미 최적의 **wsrep_slave_threads** 값이 있는 경우 가능한 연결 문제를 조사하는 동안 클러스터에서 노드를 제외할 수 있습니다.

8장. 리소스 문제 해결

리소스 오류가 발생하는 경우 문제의 원인 및 위치를 조사하고 실패한 리소스를 수정하고 선택적으로 리소스를 정리해야 합니다. 배포에 따라 리소스 실패의 많은 원인이 있을 수 있으며 리소스를 조사하여 문제를 해결하는 방법을 결정해야 합니다.

예를 들어 리소스 제약 조건을 확인하여 리소스가 서로 중단되지 않고 리소스가 서로 연결할 수 있는지 확인할 수 있습니다. 다른 컨트롤러 노드보다 펜싱되는 컨트롤러 노드를 검사하여 가능한 통신 문제를 식별할 수도 있습니다.

8.1. 리소스 제약 조건 보기

각 리소스가 있는 위치와 관련된 제약 조건, 리소스가 시작되는 순서 및 리소스를 다른 리소스와 함께 배치해야 하는지 여부를 포함하여 서비스 시작 방법에 대한 제약 조건을 볼 수 있습니다.

모든 리소스 제약 조건 보기

컨트롤러 노드에서 **pcs constraint show** 명령을 실행합니다.

```
$ sudo pcs constraint show
```

다음 예제는 컨트롤러 노드의 **pcs constraint show** 명령의 잘린 출력을 보여줍니다.

```
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)
```

이 출력에는 다음과 같은 주요 제약 조건 유형이 표시됩니다.

위치 제한

리소스를 할당할 수 있는 위치를 나열합니다.

- *첫 번째 제약 조건은 **galera-role** 특성이 **true** 로 설정된 노드에서 실행되도록 **galera-bundle** 리소스를 설정하는 규칙을 정의합니다.*
- *두 번째 위치 제한 조건은 **IP** 리소스 **ip-192.168.24.15** 가 **haproxy-role** 특성이 **true** 로 설정된 노드에서만 실행되도록 지정합니다. 즉, 클러스터가 서비스에 연결할 수 있도록 필요한 **haproxy** 서비스와 **IP** 주소를 연결합니다.*
- *세 번째 위치 제한 조건은 **ipmilan** 리소스가 각 컨트롤러 노드에서 비활성화되어 있음을 보여줍니다.*

순서 제한

*리소스를 시작할 수 있는 순서를 나열합니다. 이 예에서는 **HAProxy** 서비스 전에 가상 **IP** 주소 리소스 **IPaddr2** 를 시작하도록 설정하는 제약 조건을 보여줍니다.*

참고

*순서 제한 조건은 **IP** 주소 리소스 및 **HAproxy**에만 적용됩니다. **systemd**는 **Compute**와 같은 서비스가 **Galera**와 같은 종속 서비스의 중단을 완화할 것으로 예상되므로 다른 모든 리소스를 관리합니다.*

공동 배치 제한

*함께 있어야 하는 리소스를 나열합니다. 모든 가상 **IP** 주소는 **haproxy-bundle** 리소스에 연결됩니다.*

Galera 위치 제약 조건 보기

컨트롤러 노드에서 **pcs property show** 명령을 실행합니다.

```
$ sudo pcs property show
```

출력 예:

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

이 출력에서 모든 컨트롤러 노드에 대해 **galera-role** 특성이 **true** 인지 확인할 수 있습니다. 즉, **galera-bundle** 리소스는 이러한 노드에서만 실행됩니다. 동일한 개념이 다른 위치 제약 조건과 관련된 다른 속성에 적용됩니다.

8.2. 컨트롤러 노드 리소스 문제 조사

문제의 유형 및 위치에 따라 리소스를 조사하고 수정하는 데 걸릴 수 있는 다양한 접근 방식이 있습니다.

컨트롤러 노드 문제 조사

컨트롤러 노드에 대한 상태 점검이 실패하면 컨트롤러 노드 간 통신 문제가 표시될 수 있습니다. 조사하려면 컨트롤러 노드에 로그인하고 서비스가 올바르게 시작되는지 확인합니다.

개별 리소스 문제 조사

컨트롤러의 대부분의 서비스가 올바르게 실행 중인 경우 **pcs status** 명령을 실행하고 특정 서비스 장애에 대한 정보는 출력을 확인할 수 있습니다. 리소스가 실패하는 컨트롤러에 로그인하고 컨트롤러 노드에서 리소스 동작을 조사할 수도 있습니다.

절차

*다음 절차에서는 **openstack-cinder-volume** 리소스를 조사하는 방법을 보여줍니다.*

1. *리소스가 실패하는 컨트롤러 노드를 찾아 로그인합니다.*

2. ***systemctl status** 명령을 실행하여 리소스 상태 및 최근 로그 이벤트를 표시합니다.*

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


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

```
09:25:56.146 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...essfully.
Hint: Some lines were ellipsized, use -l to show in full.
```

3. *출력의 정보를 기반으로 실패한 리소스를 수정합니다.*

4. ***pcs resource cleanup** 명령을 실행하여 리소스의 상태 및 실패 횟수를 재설정합니다.*

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

9장. 고가용성 RED HAT CEPH STORAGE 클러스터 모니터링

Red Hat Ceph Storage를 사용하여 오버클라우드를 배포할 때 **Red Hat OpenStack Platform**은 **ceph-mon** 모니터 데몬을 사용하여 **Ceph** 클러스터를 관리합니다. **director**는 모든 컨트롤러 노드에 데몬을 배포합니다.

Ceph 모니터링 서비스의 상태 보기

컨트롤러 노드에서 **service ceph status** 명령을 실행하여 **Ceph** 모니터링 서비스가 실행 중인지 확인합니다.

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

Ceph 모니터링 구성 보기

컨트롤러 노드 또는 **Ceph** 노드에서 **/etc/ceph/ceph.conf** 파일을 열어 모니터링 구성 매개변수를 확인합니다.

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

이 예에서는 다음 정보를 보여줍니다.

- 세 개의 컨트롤러 노드는 모두 **mon_initial_members** 매개변수를 사용하여 **Red Hat Ceph Storage** 클러스터를 모니터링하도록 구성됩니다.
- **172.19.0.11/24** 네트워크는 컨트롤러 노드와 **Red Hat Ceph Storage** 노드 간에 통신 경로를 제공하도록 구성되어 있습니다.

- ***Red Hat Ceph Storage** 노드는 컨트롤러 노드에서 별도의 네트워크에 할당되며 모니터링 컨트롤러 노드의 **IP** 주소는 **172.18.0.15,172.18.0.16** 및 **172.18.0.17** 입니다.*

*개별 **Ceph** 노드 상태 보기*

***Ceph** 노드에 로그인하고 **ceph -s** 명령을 실행합니다.*

```
# 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
```

*이 예제 출력에서는 상태 매개변수 값이 **HEALTH_OK** 임을 보여줍니다. 이는 **Ceph** 노드가 활성 상태이고 정상임을 나타냅니다. 또한 **3**개의 **overcloud-controller** 노드에서 실행 중인 **3**개의 **Ceph** 모니터 서비스와 서비스의 **IP** 주소 및 포트도 표시됩니다.*

***Red Hat Ceph Storage**에 대한 자세한 내용은 **Red Hat Ceph** 제품 페이지를 참조하십시오.*