



Red Hat OpenStack Platform 17.1

고가용성 서비스 관리

Red Hat OpenStack Platform에서 고가용성 계획, 배포 및 관리

Red Hat OpenStack Platform 17.1 고가용성 서비스 관리

Red Hat OpenStack Platform에서 고가용성 계획, 배포 및 관리

OpenStack Team
rhos-docs@redhat.com

법적 공지

Copyright © 2023 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.

초록

OpenStack 환경을 효율적으로 가동하고 실행하려면 Red Hat OpenStack Platform director를 사용하여 OpenStack의 모든 주요 서비스에 걸쳐 고가용성 및 로드 밸런싱을 제공하는 구성을 생성합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체	4
RED HAT 문서에 관한 피드백 제공	5
1장. RED HAT OPENSTACK PLATFORM 고가용성 개요 및 계획	6
1.1. RED HAT OPENSTACK PLATFORM 고가용성 서비스	6
1.2. 고가용성 하드웨어 할당 계획	7
1.3. 고가용성 네트워킹 계획	7
1.4. 고가용성 환경에 액세스	8
1.5. 추가 리소스	8
2장. 배포 예: COMPUTE 및 CEPH를 사용한 고가용성 클러스터	9
2.1. 고가용성 하드웨어 사양의 예	9
2.2. 고가용성 네트워크 사양의 예	10
2.3. 고가용성 언더클라우드 설정 파일의 예	11
2.4. 고가용성 오버클라우드 구성 파일의 예	14
2.5. 추가 리소스	20
3장. PACEMAKER를 사용하여 고가용성 서비스 관리	21
3.1. PACEMAKER 리소스 번들 및 컨테이너	21
3.2. PACEMAKER 클러스터 상태 확인	24
3.3. 고가용성 클러스터에서 번들 상태 확인	25
3.4. 고가용성 클러스터에서 가상 IP에 대한 리소스 정보 보기	26
3.5. 고가용성 클러스터에서 가상 IP에 대한 네트워크 정보 보기	27
3.6. 펜싱 에이전트 및 PACEMAKER 데몬 상태 확인	29
3.7. 추가 리소스	30
4장. STONITH를 사용하여 컨트롤러 노드 펜싱	31
4.1. 지원되는 펜싱 에이전트	31
4.2. 오버클라우드에 펜싱 배포	32
4.3. 오버클라우드에서 펜싱 테스트	35
4.4. STONITH 장치 정보 보기	36
4.5. 펜싱 매개변수	36
4.6. 추가 리소스	38
5장. HAPROXY를 사용하여 트래픽 로드 밸런싱	39
5.1. HAPROXY 작동 방식	39
5.2. HAPROXY 통계 보기	40
5.3. 추가 리소스	40
6장. 외부 로드 밸런서를 사용하도록 오버클라우드 구성	41
6.1. 외부 로드 밸런서를 위한 환경 준비	41
6.2. 외부 로드 밸런서에 대한 오버클라우드 네트워크 구성	43
6.3. 외부 로드 밸런서 환경 파일 생성	44
6.4. 외부 로드 밸런싱을 위한 SSL 구성	46
6.5. 외부 로드 밸런서를 사용하여 오버클라우드 배포	47
6.6. 추가 리소스	48
7장. 설정 예: 외부 HAPROXY 로드 밸런서가 있는 OVERCLOUD	49
7.1. HAPROXY 구성 파일 예	49
7.2. 로드 밸런싱을 사용하는 서비스에 대한 구성 매개변수	55
8장. GALERA를 사용하여 데이터베이스 복제 관리	66
8.1. MARIADB 클러스터에서 호스트 이름 확인	66

8.2. MARIADB 클러스터 무결성 확인	67
8.3. MARIADB 클러스터에서 데이터베이스 노드 무결성 확인	68
8.4. MARIADB 클러스터에서 데이터베이스 복제 성능 테스트	69
8.5. 추가 리소스	71
9장. 고가용성 리소스 문제 해결	72
9.1. 고가용성 클러스터에서 리소스 제약 조건 보기	72
9.2. PACEMAKER 리소스 문제 조사	74
9.3. SYSTEMD 리소스 문제 조사	75
10장. 고가용성 RED HAT CEPH STORAGE 클러스터 모니터링	78
10.1. RED HAT CEPH 모니터링 서비스 상태 확인	78
10.2. RED HAT CEPH 모니터링 구성 확인	78
10.3. RED HAT CEPH 노드 상태 확인	79
10.4. 추가 리소스	79

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 [CTO Chris Wright의 메시지](#)를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 어떻게 더 나은지 알려주십시오.

Jira에서 문서 피드백 제공

문제 생성 양식을 사용하여 문서에 대한 피드백을 제공합니다. Jira 문제는 Red Hat OpenStack Platform Jira 프로젝트에서 생성되어 피드백의 진행 상황을 추적할 수 있습니다.

1. Jira에 로그인했는지 확인합니다. Jira 계정이 없는 경우 피드백을 제출할 계정을 생성합니다.
2. 다음 링크를 클릭하여 **문제 생성** 페이지를 엽니다.
<https://issues.redhat.com/secure/CreateInfoDetails!init.jspx?pid=12336920&summary=Documentation%20feedback:%20%3CAdd%20summary%20here%3E&i<Include+the+documentation+URL,+the%20chapter+or+section+number,+and+a+detailed+descrip>
3. **요약** 및 **설명** 필드를 작성합니다. **설명** 필드에 문서 URL, 장 또는 섹션 번호, 문제에 대한 자세한 설명을 포함합니다. 양식의 다른 필드를 수정하지 마십시오.
4. **생성**을 클릭합니다.

1장. RED HAT OPENSTACK PLATFORM 고가용성 개요 및 계획

RHOSP(Red Hat OpenStack Platform) HA(고가용성)는 배포에 대한 페일오버 및 복구를 오케스트레이션하는 서비스 컬렉션입니다. HA 배포를 계획할 때 하드웨어 할당 및 네트워크 구성과 같은 환경의 다양한 측면에 대한 고려 사항을 검토해야 합니다.

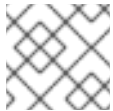
1.1. RED HAT OPENSTACK PLATFORM 고가용성 서비스

RHOSP(Red Hat OpenStack Platform)는 HA(고가용성)를 구현하는 데 필요한 서비스를 제공하기 위해 여러 기술을 사용합니다. 이러한 서비스에는 Galera, RabbitMQ, Redis, HAProxy, Pacemaker에서 관리하는 개별 서비스, Podman에서 관리하는 Systemd 및 일반 컨테이너 서비스가 포함됩니다.

1.1.1. 서비스 유형

코어 컨테이너

핵심 컨테이너 서비스는 Galera, RabbitMQ, Redis, HAProxy입니다. 이러한 서비스는 모든 컨트롤러 노드에서 실행되며 시작, 중지 및 재시작 작업에 대한 특정 관리 및 제약 조건이 필요합니다. Pacemaker를 사용하여 핵심 컨테이너 서비스를 시작, 관리 및 해결합니다.



참고

RHOSP는 [MariaDB Galera Cluster](#)를 사용하여 데이터베이스 복제를 관리합니다.

active-passive

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

systemd 및 일반 컨테이너

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

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

1.1.2. 서비스 모드

HA 서비스는 다음 모드 중 하나로 실행될 수 있습니다.

active-active

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



중요

Active-active 모드는 에지 사이트의 DCN(Distributed Compute node) 아키텍처에서만 지원됩니다.

active-passive

활성 모드에서 실행할 수 없는 서비스는 활성 모드에서 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 보냅니다. 이렇게 하면 여러 Galera 노드에서 동일한 데이터에 대한 동시 연결을 너무 많이 방지할 수 있습니다.

1.2. 고가용성 하드웨어 할당 계획

하드웨어 할당을 계획할 때 배포에서 실행할 노드 수와 컴퓨팅 노드에서 실행하려는 가상 머신(vm) 인스턴스 수를 고려하십시오.

컨트롤러 노드

대부분의 스토리지가 아닌 서비스는 컨트롤러 노드에서 실행됩니다. 모든 서비스는 세 개의 노드에 복제되며 active-active 또는 active-passive 서비스로 구성됩니다. HA(고가용성) 환경에는 최소 3개의 노드가 필요합니다.

Red Hat Ceph Storage 노드

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

컴퓨팅 노드

컴퓨팅 노드에서 실행되는 VM(가상 머신) 인스턴스입니다. 용량 요구 사항을 충족하는 데 필요한 만큼 컴퓨팅 노드를 배포하고 마이그레이션 및 재부팅 작업을 수행할 수 있습니다. VM이 다른 컴퓨팅 노드 및 공용 네트워크의 VM에 액세스할 수 있도록 컴퓨팅 노드를 스토리지 네트워크 및 프로젝트 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 Clusters](#)를 참조하십시오.

1.3. 고가용성 네트워킹 계획

가상 및 물리적 네트워크를 계획할 때 provisioning 네트워크 스위치 구성 및 외부 네트워크 스위치 구성을 고려하십시오.

네트워크 구성 외에도 다음 구성 요소를 배포해야 합니다.

프로비저닝 네트워크 스위치

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

컨트롤러/외부 네트워크 스위치

- 배포의 다른 VLAN에 대해 VLAN 태그를 수행하도록 이 스위치를 구성해야 합니다.
- 외부 네트워크에 대한 VLAN 100 트래픽만 허용합니다.

네트워킹 하드웨어 및 keystone 끝점

- 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.

1.4. 고가용성 환경에 액세스

HA(고가용성) 노드를 조사하려면 **stack** 사용자를 사용하여 오버클라우드 노드에 로그인하고 **openstack server list** 명령을 실행하여 노드의 상태 및 세부 정보를 확인합니다.

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

절차

1. 실행 중인 HA 환경에서 **stack** 사용자로 언더클라우드에 로그인합니다.
2. 오버클라우드 노드의 IP 주소를 확인합니다.

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

3. 오버클라우드 노드 중 하나에 로그인합니다.

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

<node_ip >를 로그인하려는 노드의 IP 주소로 바꿉니다.

1.5. 추가 리소스

- [2장. 배포 예: Compute 및 Ceph를 사용한 고가용성 클러스터](#)

2장. 배포 예: COMPUTE 및 CEPH를 사용한 고가용성 클러스터

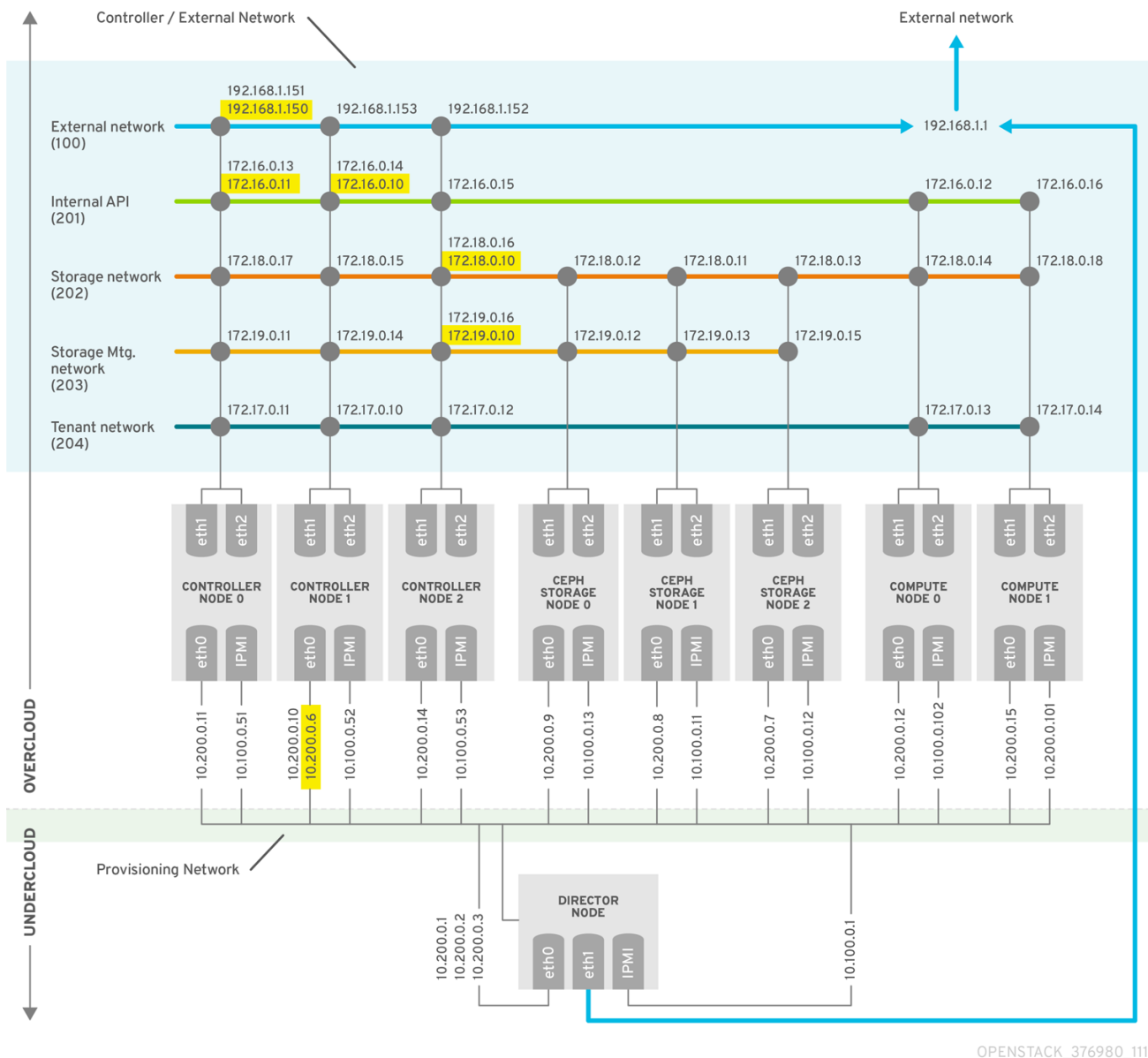
이 예에서는 OpenStack Compute 서비스 및 Red Hat Ceph Storage를 사용한 고가용성 배포를 위한 아키텍처, 하드웨어 및 네트워크 사양, 언더클라우드 및 오버클라우드 구성 파일을 보여줍니다.



중요

이 배포는 테스트 환경의 참조로 사용하기 위한 것이며 프로덕션 환경에서는 지원되지 않습니다.

그림 2.1. 고가용성 배포 아키텍처의 예



2.1. 고가용성 하드웨어 사양의 예

예제 HA 배포에서는 특정 하드웨어 구성을 사용합니다. 자체 테스트 배포에 필요에 따라 CPU, 메모리, 스토리지 또는 NIC를 조정할 수 있습니다.

표 2.1. 물리적 컴퓨터

컴퓨터 수	목적	CPU	메모리	디스크 공간	전원 관리	NIC
1	언더클라우드 노드	4	24GB	40GB	IPMI	2개의 외부 1개; 프로비저닝 시 1개(프로비저닝 시 1개) + 1개의 IPMI
3	컨트롤러 노드	4	24GB	40GB	IPMI	3(2개의 오버클라우드에 본딩; 프로비저닝 시 1개) + 1 IPMI
3	Ceph Storage 노드	4	24GB	40GB	IPMI	3(2개의 오버클라우드에 본딩; 프로비저닝 시 1개) + 1 IPMI
2	컴퓨팅 노드 (필요한 대로 추가)	4	24GB	40GB	IPMI	3(2개의 오버클라우드에 본딩; 프로비저닝 시 1개) + 1 IPMI

2.2. 고가용성 네트워크 사양의 예

예제 HA 배포에서는 특정 가상 및 물리적 네트워크 구성을 사용합니다. 자체 테스트 배포에 필요한 대로 구성을 조정할 수 있습니다.



참고

이 예에서는 컨트롤 플레인의 하드웨어 중복성과 오버클라우드 keystone 관리 엔드포인트가 구성된 프로비저닝 네트워크가 포함되지 않습니다. 고가용성 네트워킹 계획에 대한 자세한 내용은 [1.3절. "고가용성 네트워킹 계획"](#) 을 참조하십시오.

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	컨트롤러/외부 (overcloud)	해당 없음	VLAN과 NIC 결합

물리적 NIC	목적	VLAN	설명
	외부 네트워크	VLAN 100	환경 외부에서 프로젝트 네트워크, 내부 API 및 OpenStack Horizon 대시보드로 액세스할 수 있도록 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
	프로젝트 네트워크	VLAN 204	RHOSP에 프로젝트 네트워크 서비스 제공

2.3. 고가용성 언더클라우드 설정 파일의 예

예제 HA 배포에서는 언더클라우드 구성 파일 **instackenv.json**, **undercloud.conf**, **network-environment.yaml** 을 사용합니다.

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. 고가용성 오버클라우드 구성 파일의 예

예제 HA 배포에서는 오버클라우드 구성 파일 **haproxy.cfg**, **corosync.cfg**, **ceph.cfg** 를 사용합니다.

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

이 파일은 HAProxy가 관리하는 서비스를 식별합니다. HAProxy가 모니터링하는 서비스의 설정이 포함되어 있습니다. 이 파일은 모든 컨트롤러 노트에서 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
}

odelist {
    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 포함하여 Ceph 고가용성 설정이 포함되어 있습니다.

```

[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. 추가 리소스

- [director와 함께 Red Hat Ceph Storage 및 Red Hat OpenStack Platform 배포](#)
- [1장. Red Hat OpenStack Platform 고가용성 개요 및 계획](#)

3장. PACEMAKER를 사용하여 고가용성 서비스 관리

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

3.1. PACEMAKER 리소스 번들 및 컨테이너

Pacemaker는 RHOSP(Red Hat OpenStack Platform) 서비스를 번들 세트 리소스 또는 번들로 관리합니다. 이러한 서비스의 대부분은 동일한 방식으로 시작하고 각 컨트롤러 노드에서 항상 실행되는 **active-active** 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. 이를 위해서는 다른 노드에서 서비스 상태를 제어하고 설정하는 특정 리소스 에이전트가 필요합니다.



중요

- **podman** 또는 **systemctl** 을 사용하여 번들 또는 컨테이너를 관리할 수 없습니다. 명령을 사용하여 서비스 상태를 확인할 수 있지만 Pacemaker를 사용하여 이러한 서비스에 대한 작업을 수행해야 합니다.
- Podman에서 Pacemaker에서 제어하는 Podman 컨테이너에 **RestartPolicy** 가 **no** 로 설정되어 있습니다. Podman이 아닌 Pacemaker에서 컨테이너 시작 및 중지 작업을 제어하도록 하기 위한 것입니다.

3.1.1. 간단한 번들 세트 리소스(간단한 번들)

간단한 번들 세트 리소스 또는 간단한 번들은 각각 컨트롤러 노드에 배포하려는 동일한 Pacemaker 서비스를 포함하는 컨테이너 세트입니다.

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

```
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
```

각 번들에 대해 다음 세부 정보를 확인할 수 있습니다.

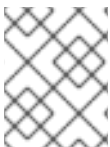
- Pacemaker에서 서비스에 할당하는 이름입니다.
- 번들과 연결된 컨테이너에 대한 참조입니다.
- 다른 컨트롤러 노드에서 실행 중인 복제본의 목록 및 상태입니다.

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

```
$ 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)
```

이 예제에서는 번들의 컨테이너에 대한 다음 정보를 보여줍니다.

- **image:** 언더클라우드의 로컬 레지스트리를 참조하는 컨테이너에서 사용하는 이미지입니다.
- **Network:** Container network type, which is **"host"** in the example.
- **options:** 컨테이너에 대한 특정 옵션
- **replicas:** 클러스터에서 실행해야 하는 컨테이너의 사본 수를 나타냅니다. 각 번들에는 각 컨트롤러 노드에 하나씩 3개의 컨테이너가 포함되어 있습니다.
- **run-command:** 컨테이너를 생성하는 데 사용되는 시스템 명령입니다.
- **Storage Mapping:** 각 호스트의 로컬 경로를 컨테이너에 매핑합니다. **haproxy** 구성은 **/etc/haproxy/haproxy.cfg** 파일 대신 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** 파일에 있습니다.



참고

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

3.1.2. 복합 번들 세트 리소스(복합 번들)

복잡한 번들 세트 리소스 또는 복잡한 번들은 간단한 번들에 포함된 기본 컨테이너 구성 외에도 리소스 구성을 지정하는 Pacemaker 서비스입니다.

이 구성은 실행되는 컨트롤러 노드에 따라 다른 상태를 가질 수 있는 서비스인 다중 상태 리소스를 관리하는 데 필요합니다.

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

```
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

```

이 출력에는 각 복잡한 번들에 대한 다음 정보가 표시됩니다.

- RabbitMQ: 세 개의 컨트롤러 노드는 간단한 번들과 유사하게 서비스의 독립 실행형 인스턴스를 실행합니다.
- Galera: 세 개의 컨트롤러 노드는 모두 동일한 제약 조건 하에서 Galera 마스터로 실행됩니다.
- Redis: **overcloud-controller-0** 컨테이너가 마스터로 실행 중인 반면 다른 두 컨트롤러 노드는 슬레이브로 실행됩니다. 각 컨테이너 유형은 제약 조건에 따라 실행될 수 있습니다.

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

```

[...]
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)
```

[...]

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 도달하는 데 필요한 IP 주소에서 수신 대기하지 못할 수 있습니다. 서비스의 IP 주소를 확인하는 방법에 대한 자세한 내용은 [3.4절. "고가용성 클러스터에서 가상 IP에 대한 리소스 정보 보기"](#)을 참조하십시오.

3.2. PACEMAKER 클러스터 상태 확인

Pacemaker가 실행 중인 모든 노드에서 Pacemaker 클러스터의 상태를 확인하고 활성 상태이고 실행 중인 리소스 수에 대한 정보를 볼 수 있습니다.

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

프로세스

- 모든 컨트롤러 노드에 **tripleo-admin** 사용자로 로그인합니다.

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

- pcs status** 명령을 실행합니다.

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

출력 예:

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

출력의 주요 섹션에서는 클러스터에 대한 다음 정보를 보여줍니다.

- **cluster name:** 클러스터의 이름입니다.
- **[NUM]** 노드가 구성된 노드 수: 클러스터에 대해 구성된 노드 수입니다.
- **[NUM]** 구성된 리소스: 클러스터에 대해 구성된 리소스 수입니다.
- **Online:** 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 [3.1절. "Pacemaker 리소스 번들 및 컨테이너"](#) 을 참조하십시오.

3.3. 고가용성 클러스터에서 번들 상태 확인

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

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

절차

다음 옵션 중 하나를 사용합니다.

- 언더클라우드 노드에 로그인하고 번들 상태를 확인합니다. 이 예제에서 **haproxy-bundle:**

```
$ sudo podman exec -it haproxy-bundle-podman-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
```

출력에 **haproxy** 프로세스가 컨테이너 내에서 실행 중임을 확인할 수 있습니다.

- 컨트롤러 노드에 로그인하고 번들 상태를 확인합니다. 이 예제에서 **haproxy:**

```
$ 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
```

3.4. 고가용성 클러스터에서 가상 IP에 대한 리소스 정보 보기

모든 VIP(가상 IP) 또는 특정 VIP의 상태를 확인하려면 관련 옵션을 사용하여 **pcs resource show** 명령을 실행합니다. 각 IPAddr2 리소스는 클라이언트가 서비스에 대한 액세스를 요청하는 데 사용하는 가상 IP 주소를 설정합니다. 해당 IP 주소가 있는 컨트롤러 노드가 실패하면 IPAddr2 리소스가 IP 주소를 다른 컨트롤러 노드에 다시 할당합니다.

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

프로세스

- 모든 컨트롤러 노드에 **tripleo-admin** 사용자로 로그인합니다.

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

- 다음 옵션 중 하나를 사용합니다.

- pcs resource show** 명령을 **--full** 옵션으로 실행하여 가상 IP를 사용하는 모든 리소스를 표시합니다.

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

출력 예:

```
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 노드가 실패하면 클러스터의 다른 컨트롤러 노드에 IP 주소가 다시 할당됩니다.

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

표 3.1. IP 주소 설명 및 할당 소스

IP 주소	설명	할당 대상
10.200.0.6	컨트롤러 가상 IP 주소	dhcp_start 및 dhcp_end 범위의 일부는 undercloud.conf 파일에서 10.200.0.5-10.200.0.24 로 설정

IP 주소	설명	할당 대상
192.168.1.150	공용 IP 주소	network-environment.yaml 파일의 ExternalAllocationPools 속성
172.16.0.10	컨트롤러 노드의 OpenStack API 서비스에 대한 액세스 제공	network-environment.yaml 파일의 InternalApiAllocationPools
172.16.0.11	컨트롤러 노드의 Redis 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
172.19.0.10	스토리지 관리에 대한 액세스 제공	network-environment.yaml 파일의 StorageMgmtAllocationPools

- VIP를 사용하는 리소스의 이름과 함께 **pcs resource show** 명령을 실행하여 특정 VIP 주소를 확인합니다. 이 예제에서는 **ip-192.168.1.150**:

```
$ 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)
```

3.5. 고가용성 클러스터에서 가상 IP에 대한 네트워크 정보 보기

특정 가상 IP(VIP)에 할당된 컨트롤러 노드의 네트워크 인터페이스 정보를 보고 특정 서비스에 대한 포트 번호 할당을 볼 수 있습니다.

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

절차

1. 보려는 IP 주소에 할당된 컨트롤러 노드에 로그인하고 네트워크 인터페이스에서 **ip addr show** 명령을 실행합니다. 이 예제에서는 **vlan100**입니다.

```
$ 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
```

2. **netstat** 명령을 실행하여 IP 주소를 청취하는 모든 프로세스를 표시합니다. 이 예 **192.168.1.150.haproxy**:

```
$ 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** 까지 사용할 수 있습니다. 이러한 프로세스에는 **sshd,mysqld,dhclient,ntpd** 가 포함됩니다.

3. HA 서비스에 대한 구성 파일을 열어 수신 대기하는 기본 포트 번호 할당 및 서비스를 확인합니다 (이 예제에서는 **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg**:

- TCP 포트 6080: **nova_novncproxy**
- TCP 포트 9696: **neutron**
- TCP port 8000: **heat_cfn**
- TCP 포트 80: **Horizon**

- TCP 포트 8776: **cinder**

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

따라서 **controller-0** 노드가 실패하면 HAProxy는 **192.168.1.150** 을 다른 컨트롤러 노드에만 다시 할당하면 다른 모든 서비스는 이미 대체 컨트롤러 노드에서 실행되고 있습니다.

3.6. 펜싱 에이전트 및 PACEMAKER 데몬 상태 확인

Pacemaker가 실행 중인 노드에서 펜싱 에이전트의 상태 및 Pacemaker 데몬의 상태를 확인하고 활성 상태이고 실행 중인 컨트롤러 노드 수에 대한 정보를 볼 수 있습니다.

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

프로세스

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

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

2. **pcs status** 명령을 실행합니다.

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

출력 예:

```
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
```

출력에는 **pcs status** 명령 출력의 다음 섹션이 표시되어 있습니다.

- **my-ipmilan-for-controller**: 각 컨트롤러 노드의 펜싱 유형(**stonith:fence_ipmilan**)과 IPMI 서비스가 중지되었는지 또는 실행 중인지를 표시합니다.
- **PCSD 상태**: 세 개의 컨트롤러 노드가 모두 현재 온라인 상태임을 표시합니다.
- **데몬 상태**: **corosync**, **pacemaker**, **pcsd** 세 개의 Pacemaker 데몬의 상태를 표시합니다. 이 예제에서는 세 가지 서비스가 모두 활성 상태이며 활성화됩니다.

3.7. 추가 리소스

- [고가용성 클러스터 구성 및 관리](#)

4장. STONITH를 사용하여 컨트롤러 노드 펜싱

펜싱은 클러스터와 클러스터 리소스를 보호하기 위해 실패한 노드를 격리하는 프로세스입니다. 펜싱이 없으면 오류가 발생한 노드가 클러스터에 데이터가 손상될 수 있습니다. director는 Pacemaker를 사용하여 고가용성 컨트롤러 노드 클러스터를 제공합니다.

Pacemaker에서는 STONITH라는 프로세스를 사용하여 실패한 노드를 펜싱합니다. STONITH는 "심각에 있는 다른 노드"의 약자입니다. STONITH는 기본적으로 비활성화되어 있으며 Pacemaker에서 클러스터의 각 노드의 전원 관리를 제어할 수 있도록 수동 구성이 필요합니다.

컨트롤러 노드가 상태 검사에 실패하면 Pacemaker 지정된 코디네이터(DC) 역할을 하는 컨트롤러 노드는 Pacemaker **stonith** 서비스를 사용하여 영향을 받는 컨트롤러 노드를 펜싱합니다.



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.

4.1. 지원되는 펜싱 에이전트

펜싱을 사용하여 고가용성 환경을 배포할 때 환경 요구 사항에 따라 펜싱 에이전트를 선택할 수 있습니다. 펜싱 에이전트를 변경하려면 **fencing.yaml** 파일에서 추가 매개 변수를 구성해야 합니다.

RHOSP(Red Hat OpenStack Platform)는 다음 펜싱 에이전트를 지원합니다.

IPMI(Intelligent Platform Management Interface)

RHOSP(Red Hat OpenStack Platform)에서 펜싱을 관리하는 데 사용하는 기본 펜싱 메커니즘입니다.

STONITH Block Device (SBD)

SBD(Storage-Based Death) 데몬은 Pacemaker 및 위치독 장치와 통합하여 펜싱이 트리거되고 기존 펜싱 메커니즘을 사용할 수 없는 경우 노드를 안정적으로 종료합니다.



중요

- SBD 펜싱은 **pacemaker_remote** 를 사용하는 원격 베어 메탈 또는 가상 머신 노드가 있는 클러스터에서는 지원되지 않으므로 배포에서 Instance HA를 사용하는 경우 지원되지 않습니다.
- 블록 스토리지 장치가 있는 **fence_sbd** 및 **sbd** Popill 펜싱은 지원되지 않습니다.
- SBD 펜싱은 호환되는 위치독 장치에서만 지원됩니다. 자세한 내용은 [RHEL High Availability Clusters - sbd 및 fence_sbd에 대한 지원 정책](#)을 참조하십시오.

fence_kdump

kdump 크래시 복구 서비스와 함께 배포에서 을 사용합니다. 이 에이전트를 선택하는 경우 덤프 파일을 저장할 충분한 디스크 공간이 있는지 확인합니다.

IPMI, **fence_rhevm** 또는 Redfish 펜싱 에이전트 외에 보조 메커니즘으로 이 에이전트를 구성할 수 있습니다. 여러 펜싱 에이전트를 구성하는 경우 두 번째 에이전트가 다음 작업을 시작하기 전에 첫 번째 에이전트가 작업을 완료할 충분한 시간을 할당해야 합니다.



중요

- RHOSP director는 펜싱 에이전트가 사용하는 전체 **kdump** 서비스의 구성이 아닌 **fence_kdump** STONITH 에이전트의 설정만 지원합니다. **kdump** 서비스 구성에 대한 자세한 내용은 [Red Hat Pacemaker 클러스터에서 fence_kdump 구성 방법](#)을 참조하십시오.
- Pacemaker 네트워크 트래픽 인터페이스에서 **ovs_bridges** 또는 **ovs_bonds** 네트워크 장치를 사용하는 경우 **fence_kdump** 가 지원되지 않습니다. **fence_kdump** 를 활성화하려면 네트워크 장치를 **linux_bond** 또는 **linux_bridge** 로 변경해야 합니다. 네트워크 인터페이스 구성에 대한 자세한 내용은 [네트워크 인터페이스 참조](#)를 참조하십시오.

Redfish

DMTF Redfish API를 지원하는 서버와 함께 배포에서 를 사용합니다. 이 에이전트를 지정하려면 **agent** 매개 변수의 값을 **fencing.yaml** 파일의 **fence_redfish** 로 변경합니다. Redfish에 대한 자세한 내용은 [DTMF 문서](#)를 참조하십시오.

다중 계층 펜싱

복잡한 펜싱 사용 사례를 지원하도록 여러 펜싱 에이전트를 구성할 수 있습니다. 예를 들어 **fence_kdump** 와 함께 IPMI 펜싱을 구성할 수 있습니다. 펜싱 에이전트의 순서에 따라 Pacemaker에서 각 메커니즘을 트리거하는 순서가 결정됩니다.

추가 리소스

- [4.2절. "오버클라우드에 펜싱 배포"](#)
- [4.3절. "오버클라우드에서 펜싱 테스트"](#)
- [4.5절. "펜싱 매개변수"](#)

4.2. 오버클라우드에 펜싱 배포

오버클라우드에 펜싱을 배포하려면 먼저 STONITH 및 Pacemaker의 상태를 검토하고 **fencing.yaml** 파일을 구성합니다. 그런 다음 오버클라우드를 배포하고 추가 매개변수를 설정합니다. 마지막으로 오버클라우드에 펜싱이 올바르게 배포되었는지 테스트합니다.

사전 요구 사항

- 배포에 적합한 펜싱 에이전트를 선택합니다. 지원되는 펜싱 에이전트 목록은 [4.1절. "지원되는 펜싱 에이전트"](#) 를 참조하십시오.
- director에 노드를 등록할 때 생성한 **nodes.json** 파일에 액세스할 수 있는지 확인합니다. 이 파일은 배포 중에 생성하는 **fencing.yaml** 파일에 필요한 입력입니다.
- **nodes.json** 파일에는 노드의 NIC(네트워크 인터페이스) 중 하나의 MAC 주소가 포함되어야 합니다. 자세한 내용은 [오버클라우드 노드 등록](#)을 참조하십시오.

프로세스

1. 각 컨트롤러 노드에 **tripleo-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 비활성화되었는지 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
```

4. 사용하려는 펜싱 에이전트에 따라 다음 옵션 중 하나를 선택합니다.

- IPMI 또는 RHV 펜싱 에이전트를 사용하는 경우 **fencing.yaml** 환경 파일을 생성합니다.

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



참고

이 명령은 **ilo** 및 **drac** 전원 관리 세부 정보를 동일한 IPMI로 변환합니다.

- STONITH Block Device (SBD), **fence_kdump** 또는 Redfish와 같은 다른 펜싱 에이전트를 사용하거나 사전 프로비저닝된 노드를 사용하는 경우 수동으로 **fencing.yaml** 파일을 생성합니다.

5. SBD 펜싱만 해당: **fencing.yaml** 파일에 다음 매개변수를 추가합니다.

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



참고

이 단계는 초기 오버클라우드 배포에만 적용됩니다. 기존 오버클라우드에서 SBD 펜싱을 활성화하는 방법에 대한 자세한 내용은 [RHEL 7 및 8에서 sbd 펜싱 활성화](#)를 참조하십시오.

6. 다중 계층 펜싱만 해당: 생성된 **fencing.yaml** 파일에 수준별 매개변수를 추가합니다.

```
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>
```

<parameter> 및 <value>를 펜싱 에이전트에 필요한 실제 매개변수 및 값으로 바꿉니다.

7. **overcloud deploy** 명령을 실행하고 배포와 관련된 **fencing.yaml** 파일 및 기타 환경 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 --ntp-server pool.ntp.org --neutron-network-type
vxlan --neutron-tunnel-types vxlan \
-e fencing.yaml
```

8. SBD 펜싱만 해당: 위치독 타이머 장치 간격을 설정하고 간격이 올바르게 설정되었는지 확인합니다.

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

검증

1. 오버클라우드에 **heat-admin** 사용자로 로그인하고 Pacemaker가 리소스 관리자로 구성되었는지 확인합니다.

```
$ 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
```

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.

2. 펜싱 리소스 속성을 확인합니다. STONITH 속성 값은 **fencing.yaml** 파일의 값과 일치해야 합니다.

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

추가 리소스

- [4.3절. "오버클라우드에서 펜싱 테스트"](#)
- [4.5절. "펜싱 매개변수"](#)
- [RHEL 고가용성의 구성 요소 탐색 - sbd 및 fence_sbd](#)

4.3. 오버클라우드에서 펜싱 테스트

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



중요

이 절차에서는 컨트롤러 노드에 대한 모든 연결을 의도적으로 삭제하므로 노드가 다시 시작됩니다.

사전 요구 사항

- 펜싱이 배포되어 오버클라우드에 실행됩니다. 펜싱을 배포하는 방법에 대한 자세한 내용은 [4.2절. "오버클라우드에 펜싱 배포"](#)를 참조하십시오.
- 컨트롤러 노드를 다시 시작할 수 있습니다.

절차

1. **stack** 사용자로 컨트롤러 노드에 로그인하고 인증 정보 파일을 가져옵니다.

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

2. **root** 사용자로 변경하고 컨트롤러 노드에 대한 모든 연결을 종료합니다.

```
$ 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. 다른 컨트롤러 노드에서 Pacemaker 로그 파일에서 펜싱 이벤트를 찾습니다.

```
$ ssh tripleo-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 후 **pcs status** 명령을 실행하여 재시작한 컨트롤러 노드가 클러스터에서 다시 실행되고 있는지 확인합니다. 출력에서 재시작한 컨트롤러 노드가 표시되면 펜싱 기능이 올바르게 작동합니다.

4.4. STONITH 장치 정보 보기

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

사전 요구 사항

- 펜싱이 배포되어 오버클라우드에 실행됩니다. 펜싱을 배포하는 방법에 대한 자세한 내용은 [4.2절](#). “오버클라우드에 펜싱 배포” 을 참조하십시오.

절차

- 컨트롤러 노드 목록과 STONITH 장치의 상태를 표시합니다.

```
$ 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)
```

이 출력에는 각 리소스에 대한 다음 정보가 표시됩니다.

- 펜싱 장치가 **fence_ipmilan** 과 같이 필요에 따라 시스템을 켜거나 끄는 데 사용하는 IPMI 전원 관리 서비스입니다.
- IPMI 인터페이스의 IP 주소(예: **10.100.0.51**).
- 로그인할 사용자 이름(예: **admin**)입니다.
- **abc** 와 같이 노드에 로그인하는 데 사용할 암호입니다.
- 각 호스트가 모니터링되는 시간(초)(예: **60s**)입니다.

4.5. 펜싱 매개변수

오버클라우드에 펜싱을 배포할 때 펜싱을 구성하는 데 필요한 매개변수를 사용하여 **fencing.yaml** 파일을 생성합니다.

다음 예제에서는 **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

펜싱 장치 및 각 장치의 매개변수를 나열합니다.

- **Agent:** Fencing 에이전트 이름입니다.
- **host_mac:** 프로비저닝 인터페이스의 소문자 또는 서버의 다른 네트워크 인터페이스의 mac 주소입니다. 이를 펜싱 장치의 고유 식별자로 사용할 수 있습니다.



중요

IPMI 인터페이스의 MAC 주소를 사용하지 마십시오.

- **params:** 펜싱 장치 매개변수 목록입니다.

펜싱 장치 매개변수

펜싱 장치 매개변수를 나열합니다. 이 예에서는 IPMI 펜싱 에이전트의 매개변수를 보여줍니다.

- **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

- **pcmk_host_list**: Pacemaker 호스트 목록입니다.

추가 리소스

- [4.2절. "오버클라우드에 펜싱 배포"](#)
- [4.1절. "지원되는 펜싱 에이전트"](#)

4.6. 추가 리소스

- ["Red Hat High Availability 클러스터에서 펜싱 구성"](#)

5장. HAPROXY를 사용하여 트래픽 로드 밸런싱

HAProxy 서비스는 고가용성 클러스터의 컨트롤러 노드에 대한 트래픽 로드 밸런싱과 로깅 및 샘플 구성을 제공합니다. **haproxy** 패키지에는 동일한 이름의 **systemd** 서비스에 해당하는 **haproxy** 데몬이 포함되어 있습니다. Pacemaker는 **haproxy-bundle**이라는 고가용성 서비스로 HAProxy 서비스를 관리합니다.

5.1. HAPROXY 작동 방식

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

다음 표는 HAProxy가 관리하는 서비스 목록을 보여줍니다.

표 5.1. 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 확인할 수 있습니다.

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

다음 예제에서는 **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 출력은 OpenStack Block Storage(cinder) 서비스에 대한 다음 정보를 보여줍니다.

- **172.16.0.10:8776:** 오버클라우드 내에서 사용할 내부 API 네트워크(VLAN201)의 가상 IP 주소와 포트입니다.
- **192.168.1.150:8776:** 오버클라우드 외부에서 API 네트워크에 액세스할 수 있는 VLAN100(외부 네트워크)의 가상 IP 주소 및 포트입니다.

- **8776**: OpenStack Block Storage(cinder) 서비스가 수신 대기 중인 포트 번호입니다.
- **server**: 컨트롤러 노드 이름 및 IP 주소입니다. HAProxy는 해당 IP 주소에 대한 요청을 서버 출력에 나열된 컨트롤러 노드 중 하나로 보낼 수 있습니다.
- **httpchk**: 컨트롤러 노드 서버에서 상태 점검을 활성화합니다.
- **fall 5**: 서비스가 오프라인 상태인지 확인하는 실패한 상태 점검 수입입니다.
- **2000년 간**: 연속된 두 개의 상태 점검 사이의 간격(밀리초)입니다.
- **Lead 2**: 서비스가 실행 중인지 확인하는 성공적인 상태 점검 수입입니다.

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

5.2. HAPROXY 통계 보기

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

director는 또한 HAProxy Stats 페이지에 도달하는 데 사용하는 **IP:Port** 주소를 설정하고 **haproxy.cfg** 파일에 정보를 저장합니다.

사전 요구 사항

- 고가용성이 배포 및 실행됩니다.

절차

1. HAProxy가 설치된 모든 컨트롤러 노드에서 **/var/lib/config-data/puppet-generated/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 auth** 행의 인증 정보를 입력하여 HAProxy 통계 페이지를 확인합니다.

5.3. 추가 리소스

- [HAProxy 1.8 문서](#)
- [openstack 서비스 부하를 분산하도록 haproxy.cfg가 올바르게 구성되었는지 어떻게 확인할 수 있습니까?](#)

6장. 외부 로드 밸런서를 사용하도록 오버클라우드 구성

RHOSP(Red Hat OpenStack Platform)에서 오버클라우드는 여러 컨트롤러 노드를 고가용성 클러스터로 사용하여 OpenStack 서비스의 운영 성능을 극대화합니다. 또한 클러스터는 OpenStack 서비스에 대한 로드 밸런싱을 제공하므로 컨트롤러 노드에 트래픽을 균등하게 배포하고 각 노드의 서버 과부하를 줄입니다.

기본적으로 오버클라우드는 HAProxy라는 오픈 소스 툴을 사용하여 부하 분산을 관리합니다. HAProxy는 OpenStack 서비스를 실행하는 컨트롤러 노드에 대한 트래픽을 로드 밸런싱합니다. **haproxy** 패키지에는 들어오는 트래픽을 수신 대기하고 로깅 기능 및 샘플 구성이 포함된 **haproxy** 데몬이 포함되어 있습니다.

오버클라우드에서는 고가용성 리소스 관리자 Pacemaker를 사용하여 HAProxy를 고가용성 서비스로 제어합니다. 즉, HAProxy는 각 컨트롤러 노드에서 실행되며 각 구성에 정의된 규칙 세트에 따라 트래픽을 배포합니다.

외부 로드 밸런서를 사용하여 이 배포를 수행할 수도 있습니다. 예를 들어 조직에서 전용 하드웨어 기반 로드 밸런서를 사용하여 컨트롤러 노드에 대한 트래픽 배포를 처리할 수 있습니다. 외부 로드 밸런서 및 오버클라우드 생성에 대한 구성을 정의하려면 다음 프로세스를 수행합니다.

1. 외부 로드 밸런서를 설치하고 구성합니다.
2. 오버클라우드를 외부 로드 밸런서와 통합하기 위해 heat 템플릿 매개변수로 오버클라우드를 구성하고 배포합니다. 이를 위해서는 로드 밸런서 및 잠재적인 노드의 IP 주소가 필요합니다.

외부 로드 밸런서를 사용하도록 오버클라우드를 구성하기 전에 오버클라우드에서 고가용성을 배포하고 실행해야 합니다.

6.1. 외부 로드 밸런서를 위한 환경 준비

외부 로드 밸런서에 대한 환경을 준비하려면 먼저 노드 정의 템플릿을 생성하고 director에 빈 노드를 등록합니다. 그런 다음 모든 노드의 하드웨어를 검사하고 노드를 프로필에 수동으로 태그합니다.

다음 워크플로우를 사용하여 환경을 준비합니다.

- 노드 정의 템플릿을 생성하고 Red Hat OpenStack Platform director에 빈 노드를 등록합니다. 노드 정의 템플릿 **instackenv.json**은 JSON 형식 파일이며 노드를 등록할 하드웨어 및 전원 관리 세부 정보를 포함합니다.
- 모든 노드의 하드웨어를 검사합니다. 이렇게 하면 모든 노드가 **manageable** 상태가 됩니다.
- 노드를 프로필에 수동으로 태그합니다. 이러한 프로필 태그는 노드와 플레이버와 일치합니다. 그러면 플레이버가 배포 역할에 할당됩니다.

절차

1. director 호스트에 **stack** 사용자로 로그인하고 director 인증 정보를 가져옵니다.

```
$ source ~/stackrc
```

2. 노드 정의 템플릿 **instackenv.json**을 생성하고 환경에 따라 다음 예제를 복사하여 편집합니다.

```
{
  "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. **stack** 사용자의 홈 디렉터리인 **/home/stack/instackenv.json** 에 파일을 저장한 다음 노드를 director에 등록하고 노드를 director에 등록합니다.

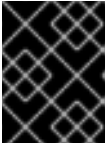
```
$ openstack overcloud node import ~/instackenv.json
```

4. 커널 및 램디스크 이미지를 모든 노드에 할당합니다.

```
$ openstack overcloud node configure
```

5. 각 노드의 하드웨어 속성을 검사합니다.

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



중요

노드는 `manageable` 상태에 있어야 합니다. 이 프로세스가 완료되었는지 확인합니다. 베어 메탈 노드의 경우 이 프로세스는 일반적으로 15분 정도 걸립니다.

6. 노드 목록을 가져와서 UUID를 확인합니다.

```
$ openstack baremetal node list
```

7. 각 노드의 **properties/capabilities** 매개변수에 `profile` 옵션을 추가하여 각 노드를 특정 프로필에 수동으로 태그합니다. 예를 들어 컨트롤러 프로필과 하나의 노드를 사용하도록 3개의 노드를 태그하여 컴퓨팅 프로필을 사용하려면 다음 명령을 사용합니다.

```
$ 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 및 **profile:control** 옵션은 노드를 각 프로필에 태그합니다.

추가 리소스

- [오버클라우드 계획](#)

6.2. 외부 로드 밸런서에 대한 오버클라우드 네트워크 구성

오버클라우드의 네트워크를 구성하려면 특정 네트워크 트래픽을 사용하도록 서비스를 분리한 다음 로컬 환경에 맞게 네트워크 환경 파일을 구성합니다. 이 파일은 오버클라우드 네트워크 환경을 설명하고 네트워크 인터페이스 구성 템플릿을 가리키며 네트워크의 서브넷 및 VLAN과 IP 주소 범위를 정의하는 `heat` 환경 파일입니다.

절차

1. 각 역할에 대한 노드 인터페이스를 구성하려면 다음 네트워크 인터페이스 템플릿을 사용자 정의합니다.
 - 각 역할에 대해 VLAN을 사용하여 단일 NIC를 구성하려면 다음 디렉터리의 예제 템플릿을 사용합니다.

-

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

- 각 역할에 대해 본딩된 NIC를 구성하려면 다음 디렉터리의 예제 템플릿을 사용합니다.

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

2. **/home/stack/network-environment.yaml** 에서 파일을 복사하고 환경에 따라 콘텐츠를 편집하여 네트워크 환경 파일을 생성합니다.

추가 리소스

- [기본 네트워크 격리](#)
- [사용자 지정 구성 가능 네트워크](#)
- [사용자 정의 네트워크 인터페이스 템플릿](#)
- [오버클라우드 네트워크](#)

6.3. 외부 로드 밸런서 환경 파일 생성

외부 로드 밸런서를 사용하여 오버클라우드를 배포하려면 필요한 구성으로 새 환경 파일을 생성합니다. 이 예제 파일에서는 오버클라우드 배포가 시작되기 전에 외부 로드 밸런서에 여러 개의 가상 IP, 격리된 각 네트워크의 하나의 가상 IP, Redis 서비스용 가상 IP가 구성됩니다. 오버클라우드 노드 NIC 설정에서 설정을 지원하는 경우 일부 가상 IP가 같을 수 있습니다.

절차

- 다음 예제 환경 파일 **external-lb.yaml** 을 사용하여 환경 파일을 생성하고 환경에 따라 콘텐츠를 편집합니다.

```
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

```

참고

- **parameter_defaults** 섹션에는 각 네트워크에 대한 VIP 및 IP 할당이 포함되어 있습니다. 이러한 설정은 로드 밸런서의 각 서비스에 대해 동일한 IP 구성과 일치해야 합니다.
- **parameter_defaults** 섹션에서는 Redis 서비스(RedisPassword)의 관리 암호를 정의하고 각 OpenStack 서비스를 특정 네트워크에 매핑하는 **ServiceNetMap** 매개변수를 포함합니다. 로드 밸런싱 구성을 사용하려면 이 서비스를 다시 매핑해야 합니다.

6.4. 외부 로드 밸런싱을 위한 SSL 구성

외부 로드 밸런서에 대해 암호화된 엔드포인트를 구성하려면 SSL이 엔드포인트에 액세스할 수 있도록 하는 추가 환경 파일을 생성한 다음, 외부 로드 밸런싱 서버에 SSL 인증서 및 키 사본을 설치합니다. 기본적으로 오버클라우드에는 암호화되지 않은 엔드포인트 서비스를 사용합니다.

사전 요구 사항

- IP 주소 또는 도메인 이름을 사용하여 공용 끝점에 액세스하는 경우 오버클라우드 배포에 포함할 다음 환경 파일 중 하나를 선택합니다.
 - DNS(Domain Name Service)를 사용하여 공용 끝점에 액세스하려면 **/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-dns.yaml** 파일을 사용합니다.
 - IP 주소로 공용 끝점에 액세스하려면 **/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-ip.yaml** 파일을 사용합니다.

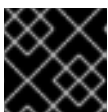
절차

1. 자체 서명된 인증서를 사용하거나 인증서 서명자가 오버클라우드 이미지의 기본 신뢰 저장소에 없는 경우 heat 템플릿 컬렉션의 **inject-trust-anchor.yaml** 환경 파일을 복사하여 오버클라우드 이미지에 인증서를 삽입합니다.

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

2. 텍스트 편집기에서 파일을 열고 루트 인증 기관 파일의 내용을 **SSLRootCertificate** 매개변수에 복사합니다.

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxGzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mIBBTFTCMBAj6rtLBKLaQblxEplzrgvp
    -----END CERTIFICATE-----
```



중요

인증 기관 콘텐츠에는 모든 새 행에 대해 동일한 들여쓰기 수준이 필요합니다.

3. **OS::TripleO::NodeTLSCADData:** 매개변수의 리소스 URL을 절대 URL로 변경합니다.

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

4. 선택 사항: DNS 호스트 이름을 사용하여 SSL/TLS를 통해 오버클라우드에 액세스하는 경우 새 환경 파일 **~/templates/cloudname.yaml** 을 생성하고 오버클라우드 끝점의 호스트 이름을 정의합니다.

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

overcloud.example.com 을 오버클라우드 엔드포인트의 DNS 호스트 이름으로 교체합니다.

6.5. 외부 로드 밸런서를 사용하여 오버클라우드 배포

외부 로드 밸런서를 사용하는 오버클라우드를 배포하려면 **openstack overcloud deploy** 를 실행하고 외부 로드 밸런서에 대한 추가 환경 파일 및 구성 파일을 포함합니다.

사전 요구 사항

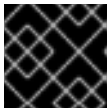
- 외부 로드 밸런서를 위한 환경이 준비되었습니다. 환경 준비 방법에 대한 자세한 내용은 다음을 참조하십시오. [6.1절. "외부 로드 밸런서를 위한 환경 준비"](#)
- 오버클라우드 네트워크는 외부 로드 밸런서용으로 구성되어 있습니다. 네트워크를 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오. [6.2절. "외부 로드 밸런서에 대한 오버클라우드 네트워크 구성"](#)
- 외부 로드 밸런서 환경 파일이 준비되었습니다. 환경 파일을 만드는 방법에 대한 자세한 내용은 다음을 참조하십시오. [6.3절. "외부 로드 밸런서 환경 파일 생성"](#)
- SSL은 외부 로드 밸런싱을 위해 구성됩니다. 외부 로드 밸런싱을 위해 SSL을 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오. [6.4절. "외부 로드 밸런싱을 위한 SSL 구성"](#)

절차

1. 외부 로드 밸런서에 대한 모든 환경 및 구성 파일을 사용하여 오버클라우드를 배포합니다.

```
$ 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>
```

대괄호 < > 의 값을 사용자 환경에 정의된 파일 경로로 바꿉니다.



중요

이 예제에 나열된 순서대로 네트워크 환경 파일을 명령에 추가해야 합니다.

이 명령에는 다음 환경 파일이 포함됩니다.

- **network-isolation.yaml**: 네트워크 격리 구성 파일
- **network-environment.yaml**: 네트워크 구성 파일
- **external-loadbalancer-vip.yaml**: 외부 로드 밸런싱 가상 IP 주소 구성 파일.
- **external-lb.yaml**: 외부 로드 밸런서 구성 파일 이 파일에 대해 다음 옵션을 설정하고 사용자 환경의 값을 조정할 수도 있습니다.

- **--control-scale 3:** 컨트롤러 노드를 3 개로 확장합니다.
- **--compute-scale 3:** 컴퓨팅 노드를 3개로 확장합니다.
- **--control-flavor control:** 컨트롤러 노드에 특정 플레이버를 사용합니다.
- **--compute-flavor compute:** 컴퓨팅 노드에 특정 플레이버를 사용합니다.
- SSL/TLS 환경 파일:
 - **SSL/TLS 끝점 환경 파일:** public endpointst에 연결하는 방법을 정의하는 환경 파일입니다. **tls-endpoints-public-dns.yaml** 또는 **tls-endpoints-public-ip.yaml** 을 사용합니다.
 - (선택 사항) **DNS 호스트 이름 환경 파일:** DNS 호스트 이름을 설정할 환경 파일입니다.
 - **루트 인증서 삽입 환경 파일:** 루트 인증 기관을 삽입하는 환경 파일입니다.

오버클라우드 배포 프로세스 중에 Red Hat OpenStack Platform director가 노드를 프로비저닝합니다. 이 프로세스를 완료하는 데 시간이 다소 걸립니다.

2. 오버클라우드 배포의 상태를 보려면 다음 명령을 입력합니다.

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

6.6. 추가 리소스

- [HAProxy를 사용하여 트래픽 로드 밸런싱](#).

7장. 설정 예: 외부 HAPROXY 로드 밸런서가 있는 OVERCLOUD

이 예제 구성은 페더레이션 HAProxy 서버를 사용하여 외부 로드 밸런싱을 제공하는 오버클라우드를 보여줍니다. 환경 요구 사항에 따라 다른 외부 로드 밸런서를 선택할 수 있습니다.

예제 구성에는 다음 요소가 포함됩니다.

- HAProxy를 실행하는 외부 로드 밸런싱 서버입니다.
- RHOSP(Red Hat OpenStack Platform) director 노드 1개
- 고가용성 클러스터 및 컴퓨팅 노드의 컨트롤러 노드 3개로 구성된 오버클라우드입니다.
- VLAN을 사용한 네트워크 격리.

이 예제에서는 각 네트워크에 대해 다음 IP 주소 할당을 사용합니다.

- 내부 API: **172.16.20.0/24**
- 테넌트: **172.16.22.0/24**
- 스토리지: **172.16.21.0/24**
- 스토리지 관리: **172.16.19.0/24**
- 외부: **172.16.23.0/24**

이러한 IP 범위에는 컨트롤러 노드의 IP 할당과 로드 밸런서가 OpenStack 서비스에 바인딩되는 가상 IP가 포함됩니다.

7.1. HAPROXY 구성 파일 예

예제 파일은 내부 HAProxy 구성 매개 변수를 보여줍니다. 샘플 구성 매개 변수를 외부 로드 밸런서를 구성하는 기준으로 사용할 수 있습니다.

HAProxy 구성 파일에는 다음 섹션이 포함되어 있습니다.

- 글로벌 구성
- 기본값 구성
- 서비스 구성

director는 컨테이너화되지 않은 환경에 대해 각 컨트롤러 노드의 `/etc/haproxy/haproxy.conf` 파일과 컨테이너화된 환경의 `/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg` 파일에 이 설정을 제공합니다.



참고

글로벌, 기본 및 서비스 매개 변수 외에도 다른 HAProxy 매개 변수도 구성해야 합니다. HAProxy 매개 변수에 대한 자세한 내용은 컨트롤러 노드의 `/usr/share/doc/haproxy-*/configuration.txt` 또는 haproxy 패키지가 설치된 모든 시스템에서 *HAProxy 구성 설명서*를 참조하십시오.

HAProxy 구성 파일 예

■

```
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-2 172.16.20.152:8042 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. 글로벌 구성 매개변수: HAProxy 구성 파일 예

글로벌 구성 매개변수 섹션에서는 로드 밸런서에 대한 프로세스 전체 매개변수 세트를 정의합니다. 구성 파일의 예제 매개변수를 사용하여 외부 로드 밸런서를 구성할 수 있습니다. 환경에 따라 매개변수 값을 조정합니다.

글로벌 구성 매개변수

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

이 예제에서는 다음 매개변수를 보여줍니다.

- **daemon:** 백그라운드 프로세스로 실행됩니다.
- **사용자 haproxy** 및 **그룹 haproxy:** 프로세스를 소유한 Linux 사용자 및 그룹을 정의합니다.
- **log:** 사용할 syslog 서버를 정의합니다.

- **maxconn**: 프로세스에 대한 최대 동시 연결 수를 설정합니다.
- **PID FILE**: 프로세스 ID에 사용할 파일을 설정합니다.

7.1.2. 기본 값 구성 매개변수: HAProxy 구성 파일 예

기본 값 구성 매개변수 섹션에서는 외부 로드 밸런서 서비스를 실행할 때 사용할 기본값 세트를 정의합니다. 구성 파일의 예제 매개변수를 사용하여 외부 로드 밸런서를 구성할 수 있습니다. 환경에 따라 매개변수 값을 조정합니다.

기본 값 구성 매개변수

```
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
```

이 예제에서는 다음 매개변수를 보여줍니다.

- **Log**: 서비스에 대한 로깅을 활성화합니다. 전역 값은 로깅 함수가 **global** 섹션의 **로그** 매개 변수를 사용한다는 것을 의미합니다.
- **mode**: 사용할 프로토콜을 정의합니다. 이 경우 기본값은 TCP입니다.
- **retry**: 연결 실패를 보고하기 전에 서버에서 수행할 재시도 횟수를 설정합니다.
- **timeout**: 특정 함수를 대기할 최대 시간을 설정합니다. 예를 들어 **timeout http-request** 는 전체 HTTP 요청을 대기할 최대 시간을 설정합니다.

7.1.3. 서비스 수준 구성 매개변수: HAProxy 구성 파일 예

서비스 수준 구성 매개변수 섹션에서는 특정 RHOSP(Red Hat OpenStack Platform) 서비스에 대한 트래픽을 로드 밸런싱할 때 사용할 매개변수 세트를 정의합니다. 구성 파일의 예제 매개변수를 사용하여 외부 로드 밸런서를 구성할 수 있습니다. 환경에 따라 매개변수 값을 조정하고 부하 분산할 각 서비스에 대한 섹션을 복사합니다.

서비스 수준 구성 매개변수

이 예에서는 **ceilometer** 서비스의 구성 매개 변수를 보여줍니다.

```
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
```

로드 밸런싱을 원하는 각 서비스는 구성 파일의 섹션에 일치해야 합니다. 각 서비스 구성에는 다음 매개변수가 포함됩니다.

- **listen**: 요청을 수신 대기하는 서비스의 이름입니다.
- **bind**: 서비스가 수신 대기하는 IP 주소 및 TCP 포트 번호입니다. 각 서비스는 다른 네트워크 트래픽 유형을 나타내는 다른 주소를 바인딩합니다.
- **server**: 서비스를 제공하는 각 서버의 이름, 서버 IP 주소 및 수신 포트, 연결 매개변수:
- **Check**: (선택 사항) 상태 점검을 활성화합니다.
- **fall 5**: (선택 사항) 5개의 실패한 상태 점검 후 서비스는 오프라인으로 간주됩니다.
- **2000년 간**: (선택 사항) 2000밀리초 또는 2초로 설정된 연속 상태 점검 간격입니다.
- **상승 2**: (선택 사항) 두 개의 성공적인 상태 점검 후 서비스가 작동하는 것으로 간주됩니다.

ceilometer 예에서 서비스는 ceilometer 서비스가 **172.16.20.2500:8777** 및 **172.16.23.250:8777** 로 제공되는 IP 주소와 포트를 식별합니다. HAProxy는 해당 주소에 대한 요청을 **overcloud-controller-0** (172.16.20.150:8777), **overcloud-controller-1** (172.16.20.151:8777) 또는 **overcloud-controller-2** (172.16.0.152:8777)로 전달합니다.

추가 리소스

- [7.2절. "로드 밸런싱을 사용하는 서비스에 대한 구성 매개변수"](#)

7.2. 로드 밸런싱을 사용하는 서비스에 대한 구성 매개변수

로드 밸런싱을 사용하는 오버클라우드의 각 서비스에 대해 다음 예제를 가이드로 사용하여 외부 로드 밸런서를 구성합니다. 환경에 따라 매개변수 값을 조정하고 부하 분산할 각 서비스에 대한 섹션을 복사합니다.



참고

대부분의 서비스에서는 기본 상태 점검 구성을 사용합니다.

- 연속된 두 개의 상태 점검 간격을 2000밀리초 또는 2초로 설정합니다.
- 두 개의 성공적인 상태 점검 후 서버는 작동으로 간주됩니다.
- 5개의 실패한 상태 점검 후 서비스는 오프라인으로 간주됩니다.

각 서비스는 해당 서비스의 **기타 정보** 섹션에 있는 기본 상태 점검 또는 추가 옵션을 나타냅니다.

Aodh

포트 번호: 8042

바인딩 대상: **internal_api**, **external**

대상 네트워크 또는 서버: **overcloud-controller-0**, **overcloud-controller-1** 및 **overcloud-controller-2**의 **internal_api**

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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**포트 번호:** 8777**바인딩 대상:** internal_api, external**대상 네트워크 또는 서버:** overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api**기타 정보:**

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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**포트 번호:** 8776**바인딩 대상:** internal_api, external**대상 네트워크 또는 서버:** overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api**기타 정보:**

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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**포트 번호:** 9292

바인딩 대상: 스토리지, 외부

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 스토리지

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 9191

바인딩 대상: internal_api

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8041

바인딩 대상: internal_api, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8004

바인딩 대상: **internal_api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.
- 이 서비스는 기본 TCP 모드 대신 HTTP 모드를 사용합니다.

HAProxy 예:

```
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

포트 번호: 8000

바인딩 대상: **internal_api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8003

바인딩 대상: **internal_api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 80

바인딩 대상: internal_api, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.
- 이 서비스는 기본 TCP 모드 대신 HTTP 모드를 사용합니다.
- 이 서비스는 UI와의 상호 작용을 위해 쿠키 기반 지속성을 사용합니다.

HAProxy 예:

```
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

포트 번호: 35357

바인딩 대상: internal_api, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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**포트 번호:** 22**바인딩 대상:** internal_api**대상 네트워크 또는 서버:** overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api**기타 정보:**

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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**포트 번호:** 5000**바인딩 대상:** internal_api, external**대상 네트워크 또는 서버:** overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api**기타 정보:**

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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**포트 번호:** 3306

바인딩 대상: internal_api

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다. 그러나 상태 점검에서는 9200 포트를 사용합니다.
- 이 서비스는 한 번에 하나의 서버에만 부하를 분산합니다.
- 각 서버는 백업이 아닌 다른 모든 서버를 사용할 수 없는 경우에만 부하 분산에서 사용됩니다.
- 서버가 오프라인 상태이면 모든 연결이 즉시 종료됩니다.
- 양쪽에서 TCP keepalive 패킷 전송을 활성화해야 합니다.
- 서버 상태를 확인하려면 HTTP 프로토콜을 활성화해야 합니다.
- 지속성을 유지하기 위해 IP 주소를 저장하도록 고정 테이블을 구성할 수 있습니다.



중요

mysql 서비스는 Galera를 사용하여 고가용성 데이터베이스 클러스터를 제공합니다. Galera는 활성-활성 구성을 지원하지만 잠금 경합을 방지하려면 로드 밸런서에 의해 적용되는 활성-패시브 구성을 사용해야 합니다.

HAProxy 예:

```
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

포트 번호: 9696

바인딩 대상: internal_api, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8773

바인딩 대상: internal_api, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8775

바인딩 대상: internal_api

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 6080

바인딩 대상: **internal _api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.
- 기본 로드 밸런싱 방법은 라운드 로빈입니다. 그러나 이 서비스에는 소스 방법을 사용합니다. 이 방법은 소스 IP 주소를 해시하고 실행 중인 서버의 총 가중치로 나눕니다. 이 방법은 또한 요청을 수신하는 서버를 지정하고 서버가 다운되거나 가동되지 않는 한 동일한 클라이언트 IP 주소가 항상 동일한 서버에 도달하도록 합니다. 실행 중인 서버 수가 변경되어 해시 결과가 변경되면 로드 밸런서에서 클라이언트를 다른 서버로 리디렉션합니다.

HAProxy 예:

```
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

포트 번호: 8774

바인딩 대상: **internal _api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8778

바인딩 대상: **internal _api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 8779

바인딩 대상: **internal_api**, external

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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

포트 번호: 6379

바인딩 대상: **internal_api**(redis 서비스 IP)

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 internal_api

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.
- **tcp-check** send/expect 시퀀스를 사용하여 상태 점검을 수행합니다. 보낼 문자열은 **info\replication\r\n** 이고 응답은 **role:master** 입니다.
- Redis 서비스는 인증을 위해 암호를 사용합니다. 예를 들어 HAProxy 구성은 AUTH 메서드 및 Redis 관리 암호를 사용하여 **tcp-check** 를 사용합니다. director는 일반적으로 임의의 암호를 생성하지만 사용자 지정 Redis 암호를 정의할 수 있습니다.
- 기본 분산 방법은 **라운드 로빈** 입니다. 그러나 이 서비스에는 **첫 번째** 방법을 사용합니다. 이렇게 하면 사용 가능한 연결 슬롯이 있는 첫 번째 서버가 연결을 수신하게 됩니다.

HAProxy 예:

```
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

포트 번호: 8080

바인딩 대상: 스토리지, 외부

대상 네트워크 또는 서버: overcloud-controller-0, overcloud-controller-1 및 overcloud-controller-2의 스토리지

기타 정보:

- 각 대상 서버는 기본 상태 점검을 사용합니다.

HAProxy 예:

```
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
```

8장. GALERA를 사용하여 데이터베이스 복제 관리

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

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.

8.1. MARIADB 클러스터에서 호스트 이름 확인

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 director가 설정한 암호를 사용합니다.

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

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 podman ps | grep -i galera
a403d96c5026 undercloudctlplane.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. 각 노드의 MariaDB 데이터베이스에서 쓰기 세트 복제 정보를 가져옵니다.

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_%';"
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| wsrep_applier_thread_count | 1      |
| wsrep_apply_oooe    | 0.018672 |
| wsrep_apply_ool     | 0.000630 |
```

```
| wsrep_apply_window | 1.021942 |
| ... | ... |
+-----+-----+
```

각 관련 변수는 **wsrep** 접두사를 사용합니다.

4. 클러스터가 올바른 노드 수를 보고하는지 확인하여 MariaDB Galera 클러스터의 상태 및 무결성을 확인합니다.

8.2. MARIADB 클러스터 무결성 확인

MariaDB Galera Cluster의 문제를 조사하려면 각 컨트롤러 노드에서 특정 **wsrep** 데이터베이스 변수를 확인하여 전체 클러스터의 무결성을 확인합니다.

절차

- 다음 명령을 실행하고 `<variable>`를 확인할 **wsrep** 데이터베이스 변수로 바꿉니다.

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

다음 예제에서는 노드의 클러스터 상태 UUID를 확인하는 방법을 보여줍니다.

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

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

표 8.1. 클러스터 무결성을 확인하는 데이터베이스 변수

Variable	요약	설명
wsrep_cluster_state_uuid	클러스터 상태 UUID	노드가 속한 클러스터의 ID입니다. 모든 노드에는 동일한 클러스터 ID가 있어야 합니다. 다른 ID가 있는 노드가 클러스터에 연결되어 있지 않습니다.
wsrep_cluster_size	클러스터의 노드 수	모든 노드에서 이를 확인할 수 있습니다. 값이 실제 노드 수보다 작으면 일부 노드가 실패하거나 연결이 끊어졌습니다.

Variable	요약	설명
wsrep_cluster_conf_id	총 클러스터 변경 수	클러스터가 여러 구성 요소 또는 파티션으로 분할되었는지 여부를 결정합니다. 파티션은 일반적으로 네트워크 오류로 인해 발생합니다. 모든 노드에는 동일한 값이 있어야 합니다. 일부 노드에서 다른 wsrep_cluster_conf_id 를 보고하는 경우 wsrep_cluster_status 값을 확인하여 노드가 클러스터에 계속 쓸 수 있는지 확인합니다(기본).
wsrep_cluster_status	기본 구성 요소 상태	노드가 클러스터에 쓸 수 있는지 여부를 결정합니다. 노드가 클러스터에 쓸 수 있는 경우 wsrep_cluster_status 값은 Primary 입니다. 다른 모든 값은 노드가 작동하지 않는 파티션의 일부를 나타냅니다.

8.3. MARIADB 클러스터에서 데이터베이스 노드 무결성 확인

MariaDB Galera Cluster에서 특정 컨트롤러 노드의 문제를 조사하려면 특정 **wsrep** 데이터베이스 변수를 확인하여 노드의 무결성을 확인합니다.

절차

- 다음 명령을 실행하고 `<variable>` 을 확인할 **wsrep** 데이터베이스 변수로 바꿉니다.

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

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

표 8.2. 노드 무결성을 확인하는 데이터베이스 변수

Variable	요약	설명
wsrep_ready	쿼리를 허용하는 노드 기능	노드에서 클러스터에서 쓰기 세트를 허용할 수 있는지 여부를 나타냅니다. 이 경우 wsrep_ready 가 ON 입니다.

Variable	요약	설명
wsrep_connected	노드 네트워크 연결	노드가 네트워크의 다른 노드에 연결할 수 있는지 여부를 나타냅니다. 이 경우 wsrep_connected 가 ON 입니다.
wsrep_local_state_comment	노드 상태	노드 상태를 요약합니다. 노드가 클러스터에 쓸 수 있는 경우 wsrep_local_state_comment 에 대한 일반적인 값은 JOIN , SST 대기, 참여, 동기화 또는 도어 일 수 있습니다. 노드가 작동하지 않는 구성 요소의 일부인 경우 wsrep_local_state_comment 값은 Initialized 입니다.



참고

- 노드가 클러스터의 노드 하위 집합에만 연결되어 있어도 **wsrep_connected** 값이 **ON** 일 수 있습니다. 예를 들어 클러스터 파티션의 경우 노드가 클러스터에 쓸 수 없는 구성 요소의 일부일 수 있습니다. 클러스터 무결성 확인에 대한 자세한 내용은 8.2절. "[MariaDB 클러스터 무결성 확인](#)" 을 참조하십시오.
- **wsrep_connected** 값이 **OFF** 이면 노드가 클러스터 구성 요소에 연결되지 않습니다.

8.4. MARIADB 클러스터에서 데이터베이스 복제 성능 테스트

MariaDB Galera Cluster의 성능을 확인하려면 특정 **wsrep** 데이터베이스 변수를 확인하여 클러스터의 복제 처리량에 대한 벤치마크 테스트를 실행합니다.

이러한 변수 중 하나를 쿼리할 때마다 **CryostatUSH STATUS** 명령은 변수 값을 재설정합니다. 벤치마크 테스트를 실행하려면 여러 쿼리를 실행하고 분산을 분석해야 합니다. 이러한 분산은 클러스터 성능에 영향을 미치는 흐름 제어의 양을 결정하는 데 도움이 될 수 있습니다.

흐름 제어는 클러스터가 복제를 관리하는 데 사용하는 메커니즘입니다. 로컬 수신 큐가 특정 임계값을 초과하면 흐름 제어는 큐 크기가 중단될 때까지 복제를 일시 중지합니다. 흐름 제어에 대한 자세한 내용은 [Galera Cluster](#) 웹 사이트의 [흐름 제어](#)를 참조하십시오.

절차

- 다음 명령을 실행하고 `<variable>`를 확인할 **wsrep** 데이터베이스 변수로 바꿉니다.

```
$ sudo podman exec galera-bundle-podman-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 8.3. 데이터베이스 복제 성능을 확인하는 데이터베이스 변수

Variable	요약	사용법
wsrep_local_recv_queue_avg	마지막 쿼리 후 로컬 수신된 쓰기 세트 대기열의 평균 크기입니다.	0.0 보다 큰 값은 복제 제한을 트리거하는 쓰기 세트를 수신하는 것처럼 노드가 쓰기 세트를 빠르게 적용할 수 없음을 나타냅니다. 이 벤치마크에 대한 자세한 내용은 wsrep_local_recv_queue_min 및 wsrep_local_recv_queue_max 를 확인합니다.
wsrep_local_send_queue_avg	마지막 쿼리 후 평균 전송 큐 길이입니다.	0.0 보다 높은 값은 복제 제한 및 네트워크 처리량 문제가 발생할 가능성이 높습니다.
wsrep_local_recv_queue_min and 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 크면 이러한 변수를 확인하여 큐 크기의 범위를 확인할 수 있습니다.
wsrep_flow_control_paused	마지막 쿼리 후 Flow Control이 노드를 일시 중지한 시간입니다.	0.0 보다 큰 값은 Flow Control이 노드를 일시 중지했음을 나타냅니다. 일시 중지 기간을 확인하려면 wsrep_flow_control_paused 값을 쿼리 사이의 초 수와 곱합니다. 최적의 값은 가능한 한 0.0 에 가깝습니다. 예를 들면 다음과 같습니다. wsrep_flow_control_paused 값이 마지막 쿼리 후 0.50 1분이면 Flow Control이 30초 동안 노드를 일시 중지했습니다. wsrep_flow_control_paused 값이 마지막 쿼리 후 1분 후 1.0 이면 Flow Control이 전체 분 동안 노드를 일시 중지했습니다.

Variable	요약	사용법
wsrep_cert_deps_distance	병렬로 적용할 수 있는 가장 낮은 시퀀스 번호(seqno) 값의 평균 차이	제한 및 일시 중지의 경우 이 변수는 평균적으로 병렬로 적용할 수 있는 쓰기 세트 수를 나타냅니다. wsrep_slave_threads 변수와 값을 비교하여 실제로 동시에 적용할 수 있는 쓰기 세트 수를 확인합니다.
wsrep_slave_threads	동시에 적용할 수 있는 스레드 수	이 변수의 값을 늘리면 더 많은 스레드를 동시에 적용할 수 있으므로 wsrep_cert_deps_distance 의 값도 늘릴 수 있습니다. wsrep_slave_threads 값은 노드의 CPU 코어 수보다 크지 않아야 합니다. 예를 들어 wsrep_cert_deps_distance 값이 20 이면 wsrep_slave_threads 값을 2 에서 4 로 늘어 노드가 적용할 수 있는 쓰기 세트의 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.

8.5. 추가 리소스

- [MariaDB Galera Cluster란 무엇입니까?](#)

9장. 고가용성 리소스 문제 해결

리소스 실패의 경우 문제의 원인 및 위치를 조사하고 실패한 리소스를 수정하고 선택적으로 리소스를 정리해야 합니다. 배포에 따라 리소스 오류가 발생할 수 있는 원인은 여러 가지가 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 더 자주 펜싱되는 컨트롤러 노드를 검사하여 가능한 통신 문제를 식별할 수도 있습니다.

리소스 문제의 위치에 따라 다음 옵션 중 하나를 선택합니다.

컨트롤러 노드 문제

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

개별 리소스 문제

컨트롤러의 대부분의 서비스가 올바르게 실행 중인 경우 **pcs status** 명령을 실행하고 특정 Pacemaker 리소스 오류에 대한 정보의 출력을 확인하거나 **systemctl** 명령을 실행하여 비Pacemaker 리소스 오류를 조사할 수 있습니다.

9.1. 고가용성 클러스터에서 리소스 제약 조건 보기

리소스 문제를 조사하기 전에 각 리소스가 있는 위치, 리소스가 시작되는 순서 및 리소스를 다른 리소스와 함께 배치해야 하는지 여부를 포함하여 서비스 시작 방법에 대한 제약 조건을 확인할 수 있습니다.

절차

- 다음 옵션 중 하나를 사용합니다.
 - 모든 리소스 제약 조건을 보려면 컨트롤러 노드에 로그인하고 **pcs constraint show** 명령을 실행합니다.

```
$ sudo pcs constraint show
```

다음 예제는 컨트롤러 노드의 **pcs constraint show** 명령에서 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 기본 제약 조건 유형이 표시됩니다.

위치 제한

리소스를 할당할 수 있는 위치를 나열합니다.

- 첫 번째 제약 조건은 **galera-bundle** 리소스가 **galera-role** 속성이 **true** 로 설정된 노드에서 실행되도록 규칙을 정의합니다.
- 두 번째 위치 제약 조건은 IP 리소스 **ip-192.168.24.15** 가 **haproxy-role** 속성이 **true** 로 설정된 노드에서만 실행되도록 지정합니다. 즉, 클러스터에서 서비스에 연결할 수 있도록 하는 데 필요한 **haproxy** 서비스와 IP 주소를 연결합니다.
- 세 번째 위치 제한 조건은 각 컨트롤러 노드에서 **ipmilan** 리소스가 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 HAProxy에만 적용됩니다. Compute와 같은 서비스는 Galera와 같은 종속 서비스의 종단을 건드릴 것으로 예상되므로 다른 모든 리소스를 관리합니다.

공동 배치 제한

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

- 특정 리소스의 제약 조건을 보려면 모든 컨트롤러 노드에 로그인하고 **pcs property show** 명령을 실행합니다.

```
$ sudo pcs property show
```

출력 예:

```
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
```

이 출력에서 리소스 제약 조건이 올바르게 설정되었는지 확인할 수 있습니다. 예를 들어 **galera-role** 속성은 모든 컨트롤러 노드에 적용됩니다. 즉, **galera-bundle** 리소스는 이러한 노드에서만 실행됩니다.

9.2. PACEMAKER 리소스 문제 조사

Pacemaker에서 관리하는 실패한 리소스를 조사하려면 리소스가 실패하는 컨트롤러 노드에 로그인하고 리소스의 상태 및 로그 이벤트를 확인합니다. 예를 들어 **openstack-cinder-volume** 리소스의 상태 및 로그 이벤트를 조사합니다.

사전 요구 사항

- **Pacemaker** 서비스가 있는 컨트롤러 노드
- 로그 이벤트를 볼 수 있는 **root** 사용자 권한

절차

1. 리소스가 실패하는 컨트롤러 노드에 로그인합니다.
2. **grep** 옵션과 함께 **pcs status** 명령을 실행하여 서비스 상태를 가져옵니다.

```
# 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.

리소스의 로그 이벤트를 확인합니다.

```
# sudo less /var/log/containers/stdouts/openstack-cinder-volume.log
[...]
2021-04-12T12:32:17.607179705+00:00 stderr F ++ cat /run_command
2021-04-12T12:32:17.609648533+00:00 stderr F + CMD='/usr/bin/cinder-volume --config-file
/usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf'
2021-04-12T12:32:17.609648533+00:00 stderr F + ARGS=
2021-04-12T12:32:17.609648533+00:00 stderr F + [[ ! -n " ]]
2021-04-12T12:32:17.609648533+00:00 stderr F + . kolla_extend_start
2021-04-12T12:32:17.611214130+00:00 stderr F +++ stat -c %U:%G /var/lib/cinder
2021-04-12T12:32:17.616637578+00:00 stderr F ++ [[ cinder:kolla != \c\i\n\d\e\r:\k\o\l\l\a ]]
2021-04-12T12:32:17.616722778+00:00 stderr F + echo 'Running command:
"/usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-dist.conf --config-file
/etc/cinder/cinder.conf"'
2021-04-12T12:32:17.616751172+00:00 stdout F Running command: '/usr/bin/cinder-volume
--config-file /usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf'
2021-04-12T12:32:17.616775368+00:00 stderr F + exec /usr/bin/cinder-volume --config-file
/usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf
```

4.

출력 및 로그의 정보를 기반으로 실패한 리소스를 수정합니다.

5.

pcs resource cleanup 명령을 실행하여 리소스의 상태 및 실패 수를 재설정합니다.

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

9.3. SYSTEMD 리소스 문제 조사

systemd가 관리하는 실패한 리소스를 조사하려면 리소스가 실패하는 컨트롤러 노드에 로그인하고 리소스의 상태 및 로그 이벤트를 확인합니다. 예를 들어 `tripleo_nova_conductor` 리소스의 상태 및 로그 이벤트를 조사합니다.

사전 요구 사항

- **systemd** 서비스가 있는 컨트롤러 노드
- 로그 이벤트를 볼 수 있는 **root** 사용자 권한

절차

1.

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

```
[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.
```

2.

리소스의 로그 이벤트를 확인합니다.

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

3.

출력 및 로그의 정보를 기반으로 실패한 리소스를 수정합니다.

4.

리소스를 다시 시작하고 서비스 상태를 확인합니다.

```
# 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
```

```
0d6583beb20508e6bacccd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -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.
```

10장. 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.

10.1. RED HAT CEPH 모니터링 서비스 상태 확인

Red Hat Ceph Storage 모니터링 서비스의 상태를 확인하려면 컨트롤러 노드에 로그인하고 **service ceph status** 명령을 실행합니다.

절차

- 컨트롤러 노드에 로그인하고 **Ceph** 모니터링 서비스가 실행 중인지 확인합니다.

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

10.2. RED HAT CEPH 모니터링 구성 확인

Red Hat Ceph Storage 모니터링 서비스의 구성을 확인하려면 컨트롤러 노드 또는 **Red Hat Ceph** 노드에 로그인하고 **/etc/ceph/ceph.conf** 파일을 엽니다.

절차

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

```
[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
```

이 예제에서는 다음 정보를 보여줍니다.

- 3개의 컨트롤러 노드 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 간의 통신 경로를 제공하도록 구성되어 있습니다.
- **Red Hat Ceph Storage** 노드는 컨트롤러 노드와 별도의 네트워크에 할당되며 모니터링 컨트롤러 노드의 IP 주소는 **172.18.0.15, 172.18.0.16, 172.18.0.17** 입니다.

10.3. RED HAT CEPH 노드 상태 확인

특정 **Red Hat Ceph Storage** 노드의 상태를 확인하려면 노드에 로그인하고 **ceph -s** 명령을 실행합니다.

절차

- **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 출력은 **health** 매개변수 값이 **Ceph** 노드가 활성 상태이고 정상임을 나타내는 **HEALTH_OK** 임을 보여줍니다. 출력에는 세 개의 **overcloud-controller** 노드와 서비스의 IP 주소 및 포트에서 실행 중인 3개의 **Ceph** 모니터 서비스도 표시됩니다.

10.4. 추가 리소스



[Red Hat Ceph 제품 페이지](#)