Red Hat OpenStack Platform 13

Director 설치 및 사용

Red Hat OpenStack Platform director를 사용하여 OpenStack 클라우드 환경 생성

Last Updated: 2023-08-25

Red Hat OpenStack Platform 13 Director 설치 및 사용

Red Hat OpenStack Platform director를 사용하여 OpenStack 클라우드 환경 생성

OpenStack Team
rhos-docs@redhat.com

법적 공지

초록

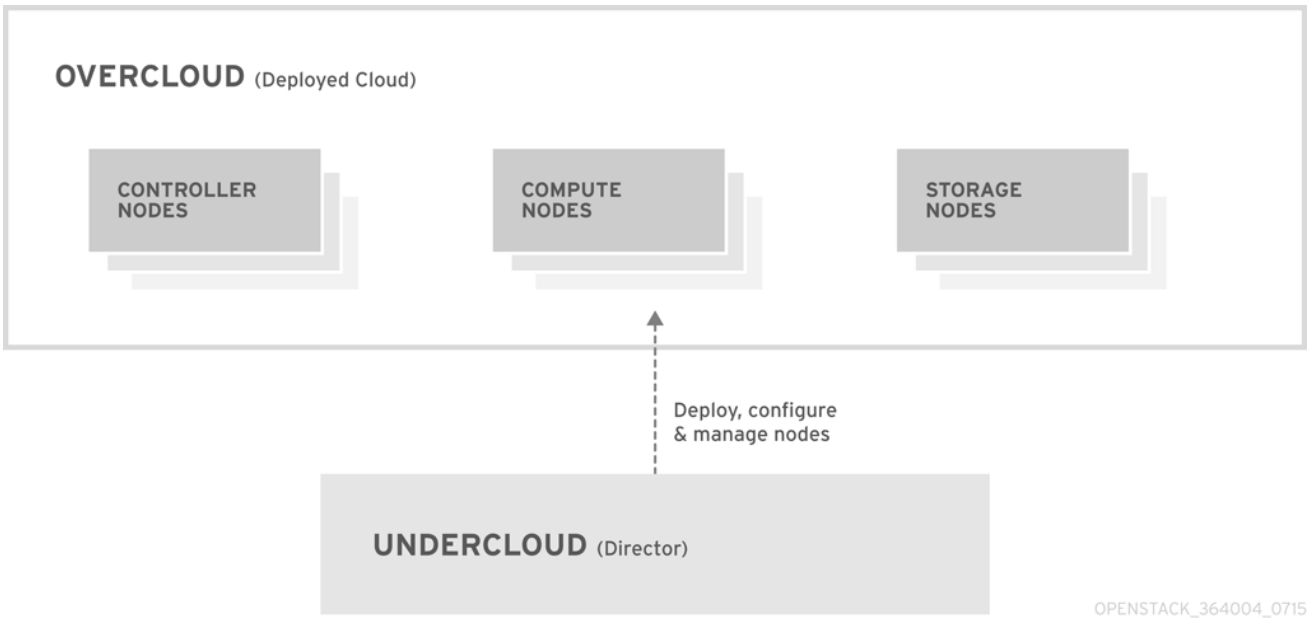
이 가이드에서는 Red Hat OpenStack Platform director를 사용하여 엔터프라이즈 환경에 Red Hat OpenStack Platform 13을 설치하는 방법을 설명합니다. 여기에는 director 설치, 환경 계획, director를 사용한 OpenStack 환경 구축 작업이 포함됩니다.

차례

1장. 소개

RHOSP(Red Hat OpenStack Platform) director는 완전한 RHOSP 환경을 설치하고 관리하기 위한 툴셋입니다. 이 프로젝트는 "OpenStack-On-OpenStack"의 약어인 OpenStack 프로젝트 TripleO를 주로 기반으로 합니다. 이 프로젝트는 OpenStack 구성 요소를 활용하여 완전히 작동하는 OpenStack 환경을 설치합니다. 여기에는 OpenStack 노드로 사용할 베어 메탈 시스템을 프로비저닝하고 제어하는 새로운 OpenStack 구성 요소가 포함됩니다.

director는 언더클라우드(undercloud)와 오버클라우드(overcloud)의 두 가지 주요 개념을 사용합니다. 먼저 언더클라우드를 설치한 다음 언더클라우드를 툴로 사용하여 오버클라우드를 설치 및 구성합니다.



1.1. 언더클라우드

언더클라우드는 Red Hat OpenStack Platform director 툴셋이 포함된 주요 관리 노드로, OpenStack을 설치한 단일 시스템에 OpenStack 환경(오버클라우드)을 구성하는 OpenStack 노드를 프로비저닝하고 관리하기 위한 구성 요소가 포함되어 있습니다. 언더클라우드의 구성 요소는 다음과 같은 여러 기능을 제공합니다.

환경 플래닝

언더클라우드에는 특정 노드 역할을 생성하고 할당하는 데 사용할 수 있는 계획 기능이 포함되어 있습니다. 언더클라우드에는 특정 노드에 할당할 수 있는 기본 노드 역할 세트가 포함되어 있습니다. 컴퓨팅, 컨트롤러 및 다양한 스토리지 역할. 사용자 지정 역할을 설정할 수도 있습니다. 또한 각 노드 역할에 포함할 Red Hat OpenStack Platform 서비스를 선택할 수 있으므로 새로운 노드 유형을 모델링하거나 특정 구성 요소를 해당 호스트에 분리하는 것이 가능합니다.

베어 메탈 시스템 컨트롤

언더클라우드는 전원 관리 컨트롤 및 PXE 기반 서비스에서 하드웨어 속성을 검색하고 OpenStack을 각 노드에 설치하는 데 각 노드의 대역 외 관리 인터페이스(일반적으로 IPMI)를 사용합니다. 이를 통해 베어 메탈 시스템을 OpenStack 노드로 프로비저닝할 수 있습니다. 전원 관리 드라이버의 전체 목록은 부록 B. *전원 관리 드라이버* 을 참조하십시오.

오케스트레이션

언더클라우드는 환경의 계획 세트 역할을 하는 YAML 템플릿 세트를 제공합니다. 언더클라우드는 이러한 환경 계획을 가져와서 해당 지침에 따라 원하는 OpenStack 환경을 생성합니다. 플랜에는 환경 생성 프로세스 중 특정 시점으로 사용자 정의를 통합할 수 있는 후크도 포함되어 있습니다.

명령줄 도구 및 웹 UI

Red Hat OpenStack Platform director는 터미널 기반 명령줄 인터페이스 또는 웹 기반 사용자 인터페이스를 통해 이러한 언더클라우드 기능을 수행합니다.

언더클라우드 구성 요소

언더클라우드는 OpenStack 구성 요소를 기본 툴셋으로 사용합니다. 여기에는 다음 구성 요소가 포함됩니다.

- OpenStack Identity(keystone) - director의 구성 요소에 인증 및 권한 부여를 제공합니다.
- OpenStack Bare Metal(ironic) 및 OpenStack Compute(nova) - 베어 메탈 노드를 관리합니다.
- OpenStack Networking(neutron) 및 Open vSwitch - 베어 메탈 노드에 대한 네트워킹을 제어합니다.
- OpenStack Image Service(glance) - 베어 메탈 머신에 기록된 이미지를 저장합니다.
- OpenStack Orchestration(heat) 및 Puppet - director에서 오버클라우드 이미지를 디스크에 기록한 후 노드의 오케스트레이션 및 노드 설정을 제공합니다.
- OpenStack Telemetry(ceilometer) - 모니터링 및 데이터 수집을 수행합니다. 여기에는 다음이 포함됩니다.
  - OpenStack Telemetry Metrics(Gnocchi) - 메트릭에 시계열 데이터베이스를 제공
  - OpenStack Telemetry Alarming(aodh) - 모니터링을 위한 알람 구성 요소를 제공
  - OpenStack Telemetry Event Storage(panko) - 모니터링을 위한 이벤트 스토리지를 제공
- OpenStack Workflow Service(mistral) - 환경 계획 가져오기 및 배포하기와 같은 특정 director 관련 작업에 대한 워크플로우를 제공합니다.
- OpenStack Messaging Service(zaqar) - OpenStack Workflow Service에 메시징 서비스를 제공합니다.
- OpenStack Object Storage(swift) - 다음을 포함한 여러 OpenStack Platform 구성 요소에 오브젝트 스토리지를 제공합니다.
  - OpenStack Image Service 용 이미지 스토리지
  - OpenStack Bare Metal에 대한 인트로스펙션 데이터
  - OpenStack Workflow Service에 대한 배포 계획

1.2. 오버클라우드

오버클라우드는 언더클라우드를 사용하여 생성된 Red Hat OpenStack Platform 환경입니다. 여기에는 생성하려는 OpenStack Platform 환경에 따라 사용자가 정의한 다양한 노드 역할이 포함됩니다. 언더클라우드에는 다음과 같은 기본 오버클라우드 노드 역할이 포함되어 있습니다.

컨트롤러

OpenStack 환경에 관리, 네트워킹 및 고가용성 기능을 제공하는 노드. 이상적인 OpenStack 환경은 고가용성 클러스터에 이러한 세 가지 노드를 함께 사용하는 것이 좋습니다.
기본 컨트롤러 노드 역할은 다음 구성 요소를 지원합니다. 해당 서비스 중 일부는 기본적으로 활성화되어 있지 않습니다. 다음을 활성화하려면 해당 구성 요소 중 일부에 사용자 지정 또는 사전 패키지 환경 파일이 있어야 합니다.

- OpenStack Dashboard(horizon)
- OpenStack Identity(keystone)
- OpenStack Compute(nova) API
- OpenStack Networking(neutron)
- OpenStack Image Service(glance)
- OpenStack Block Storage(cinder)
- OpenStack Object Storage(swift)
- OpenStack Orchestration(heat)
- OpenStack Telemetry(ceilometer)
- OpenStack Telemetry Metrics(gnocchi)
- OpenStack Telemetry Alarming(aodh)
- OpenStack Telemetry Event Storage(panko)
- OpenStack Clustering(sahara)
- OpenStack Shared File Systems(manila)
- OpenStack Bare Metal(ironic)
- MariaDB
- Open vSwitch
- 고가용성 서비스를 위한 Pacemaker 및 Galera

Compute

이러한 노드는 OpenStack 환경에 컴퓨팅 리소스를 제공합니다. 더 많은 컴퓨팅 노드를 추가하여 시간 경과에 따라 환경을 확장할 수 있습니다. 기본 컴퓨팅 노드에는 다음 구성 요소가 포함됩니다.

- OpenStack Compute(nova)
- KVM/QEMU
- OpenStack Telemetry(ceilometer) 에이전트
- Open vSwitch

스토리지

OpenStack 환경에 스토리지를 제공하는 노드입니다. 여기에는 다음을 위한 노드가 포함됩니다.

- Ceph Storage 노드 - 스토리지 클러스터를 만드는 데 사용됩니다. 각 노드에는 Ceph OSD(Object Storage Daemon)가 포함됩니다. 또한 director는 Ceph Storage 노드를 배포하는 컨트롤러 노드에 Ceph Monitor를 설치합니다.
- Block Storage(cinder) - HA Controller 노드에 대한 외부 블록 스토리지로 사용됩니다. 이 노드에는 다음 구성 요소가 포함됩니다.

- OpenStack Block Storage(cinder) 볼륨
- OpenStack Telemetry(ceilometer) 에이전트
- Open vSwitch

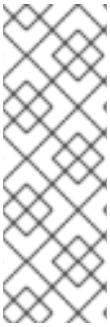
- Object Storage(swift) - 이러한 노드는 Openstack Swift에 외부 스토리지 계층을 제공합니다. 컨트롤러 노드는 Swift 프록시를 통해 이러한 노드에 액세스합니다. 이 노드에는 다음 구성 요소가 포함됩니다.
  - OpenStack Object Storage(swift) 스토리지
  - OpenStack Telemetry(ceilometer) 에이전트
  - Open vSwitch

1.3. 고가용성

Red Hat OpenStack Platform director는 컨트롤러 노드 클러스터를 사용하여 OpenStack Platform 환경에 고가용성 서비스를 제공합니다. director는 각 컨트롤러 노드에 중복된 구성 요소 세트를 설치하고 단일 서비스로 관리합니다. 이러한 유형의 클러스터 구성은 단일 컨트롤러 노드에서 작동 오류가 발생하는 경우 폴백을 제공합니다. 이로 인해 OpenStack 사용자에게 특정 수준의 연속 작업이 제공됩니다.

OpenStack Platform director는 여러 주요 소프트웨어를 사용하여 컨트롤러 노드의 구성 요소를 관리합니다.

- Pacemaker - Pacemaker는 클러스터 리소스 관리자입니다. Pacemaker는 클러스터에 있는 모든 노드에서 OpenStack 구성 요소의 가용성을 관리하고 모니터링합니다.
- HAProxy - 클러스터에 부하 분산 및 프록시 서비스를 제공합니다.
- Galera - 클러스터에 Red Hat OpenStack Platform 데이터베이스를 복제합니다.
- Memcached - 데이터베이스 캐싱을 제공합니다.



참고

- Red Hat OpenStack Platform director는 Controller 노드에서 대부분의 고가용성을 자동으로 설정합니다. 그러나 전원 관리 제어를 활성화하려면 노드에 몇 가지 수동 구성이 필요합니다. 이 안내서에는 이러한 지침이 포함되어 있습니다.
- 버전 13 이상에서는 director를 사용하여 Compute 인스턴스의 고가용성(인스턴스 HA)을 배포할 수 있습니다. 인스턴스 HA를 사용하면 해당 노드가 실패할 때 컴퓨팅 노드에서 인스턴스 비우기를 자동화할 수 있습니다.

1.4. 컨테이너화

오버클라우드의 각 OpenStack Platform 서비스는 해당 노드의 개별 Linux 컨테이너 내에서 실행됩니다. 이를 통해 서비스를 분리하고 OpenStack Platform을 쉽게 유지 관리하고 업그레이드할 수 있습니다. Red Hat은 다음을 포함하여 오버클라우드의 컨테이너 이미지를 여러 가지 방법으로 지원합니다.

- Red Hat Container Catalog에서 직접 가져오기
- 언더클라우드에서 호스트

- Satellite 6 서버에서 호스트

이 가이드에서는 레지스트리 세부 정보를 구성하고 기본적인 컨테이너 작업을 수행하는 방법에 대한 정보를 제공합니다. 컨테이너화된 서비스에 대한 자세한 내용은 컨테이너 화된 서비스로 전환 가이드를 참조하십시오.

1.5. CEPH STORAGE

OpenStack을 사용하는 대규모 조직에서는 일반적으로 수많은 클라이언트를 지원합니다. 각 OpenStack 클라이언트에는 블록 스토리지 리소스 사용에 대한 자체 요구 사항이 있습니다. glance(images), cinder(volumes) 및/또는 nova(Compute)를 단일 노드에 배포할 경우 수많은 클라이언트를 포함하는 대규모 배포에서 관리가 불가능해질 수 있습니다. OpenStack을 확장하면 이 문제가 해결됩니다.

하지만 실질적으로는 Red Hat OpenStack Platform 스토리지 계층을 수십 테라바이트에서 펩타바이트(또는 엑사바이트) 스토리지로 확장할 수 있도록 Red Hat Ceph Storage와 같은 솔루션을 사용하여 스토리지 계층을 가상화해야 하는 요구 사항도 존재합니다. Red Hat Ceph Storage는 상용 하드웨어에서 실행되는 동안 이 스토리지 가상화 계층에 고가용성 및 고성능을 제공합니다. 가상화에는 성능 저하가 나타나는 것처럼 보일 수 있지만, Ceph 스트라이프는 클러스터에서 오브젝트인 장치 이미지를 차단합니다. 이로 인해 대규모 Ceph 블록 장치 이미지가 독립 실행형 디스크보다 성능이 향상됩니다. 또한 Ceph 블록 장치는 성능 향상을 위해 캐싱, copy-on-write 복제 및 copy-on-read 복제를 지원합니다.

Red Hat Ceph Storage에 대한 자세한 내용은 Red Hat Ceph Storage 를 참조하십시오.

참고

멀티 아키텍처 클라우드의 경우 사전 설치 또는 외부 Ceph *만* 지원됩니다. 자세한 내용은 Integrating an Overcloud with an Existing Red Hat Ceph Cluster 및 부록 G. *Red Hat OpenStack Platform for POWER* 를 참조하십시오.

2장. 요구 사항

이 장에서는 director를 사용하여 Red Hat OpenStack Platform을 프로비저닝하기 위한 환경을 설정하는 주요 요구 사항에 대해 간단히 설명합니다. 여기에는 director 설정, 액세스, director에서 OpenStack 서비스에 프로비저닝하는 호스트에 대한 하드웨어 요구 사항이 포함됩니다.

참고

Red Hat OpenStack Platform을 배포하기 전에 사용 가능한 배포 방법의 특성을 고려해야 합니다. 자세한 내용은 Installing and Managing Red Hat OpenStack Platform 에서 참조하십시오.

2.1. 환경 요구 사항

최소 요구 사항:

- Red Hat OpenStack Platform director용 호스트 머신 1개
- Red Hat OpenStack Platform 컴퓨팅 노드 용 호스트 머신 1개
- Red Hat OpenStack Platform 컨트롤러 노드 용 호스트 머신 1개

권장 요구 사항:

- Red Hat OpenStack Platform director용 호스트 머신 1개
- Red Hat OpenStack Platform 컴퓨팅 노드 용 호스트 머신 3개
- 클러스터의 Red Hat OpenStack Platform Controller 노드 용 호스트 머신 3개
- 클러스터의 Red Hat Ceph Storage 노드용 호스트 머신 3개

다음을 확인합니다.

- 모든 노드에 베어 메탈 시스템을 사용하는 것이 좋습니다. 최소한 Compute 노드 및 Ceph Storage 노드에는 베어 메탈 시스템이 필요합니다.
- 모든 오버클라우드 베어 메탈 시스템에는 IPMI(Intelligent Platform Management Interface)가 필요합니다. 이는 director가 전원 관리를 제어하기 때문입니다.
- 각 노드의 내부 BIOS 시계를 UTC로 설정합니다. 이렇게 하면 시간대 오프셋을 적용하기 전에 **hwclock** 이 BIOS 시계를 동기화할 때 미래 날짜의 파일 타임스탬프 관련 문제가 발생하지 않습니다.
- Red Hat OpenStack Platform에는 로케일 설정의 일부로 특수 문자 인코딩 요구 사항이 있습니다.
  - 모든 노드에서 UTF-8 인코딩을 사용하십시오. **LANG** 환경 변수가 모든 노드에서 **en_US.UTF-8**으로 설정되어 있는지 확인하십시오.
  - Red Hat Ansible Tower를 사용하여 Red Hat OpenStack Platform 리소스를 자동으로 생성하는 경우 ASCII가 아닌 문자를 사용하지 마십시오.
- POWER(ppc64le) 하드웨어에 오버클라우드 Compute 노드를 배포하려면 부록 G. *Red Hat OpenStack Platform for POWER* 의 개요를 참조하십시오.

2.2. 언더클라우드 요구 사항

director를 호스팅하는 언더클라우드 시스템은 오버클라우드의 모든 노드에 대한 프로비저닝 및 관리를 제공합니다.

- Intel 64 또는 AMD64 CPU 확장을 지원하는 8코어 64비트 x86 프로세서입니다.
- 최소 16GB의 RAM
  - **ceph-ansible** 플레이북은 언더클라우드에서 배포된 10개 호스트당 1GB RSS(Resident Set Size)를 사용합니다. 배포된 오버클라우드에서 기존 Ceph 클러스터를 사용하거나 새 Ceph 클러스터를 배포하는 경우 언더클라우드 RAM을 적절하게 프로비저닝합니다.
- 루트 디스크에서 최소 100GB의 사용 가능한 디스크 공간이 있어야 합니다. 여기에는 다음이 포함됩니다.
  - 컨테이너 이미지의 경우 10GB
  - 10GB: 노드 프로비저닝 프로세스 중 QCOW2 이미지 변환 및 캐싱에 사용
  - 80GB 이상: 일반 용도, 로깅, 매트릭스 및 확장
- 최소 2개의 1GB의 네트워크 인터페이스 카드가 필요합니다. 그러나 특히 오버클라우드 환경에서 다수의 노드를 프로비저닝하는 경우에는 프로비저닝 네트워크 트래픽에 대비해 10Gbps 인터페이스를 사용하는 것이 좋습니다.
- 최신 버전의 Red Hat Enterprise Linux 7은 호스트 운영 체제로 설치됩니다.
- SELinux가 호스트의 **강제** 모드에서 활성화됩니다.

2.2.1. 가상화 지원

Red Hat은 다음 플랫폼에서 가상화된 언더클라우드만 지원합니다.

플랫폼	참고
Kernel-based Virtual Machine (KVM)	인증된 하이퍼바이저에 나열된 Red Hat Enterprise Linux 7에서 호스트됩니다.
Red Hat Virtualization	인증된 하이퍼바이저에 나열된 Red Hat Virtualization 4.x에서 호스트됩니다.
Microsoft Hyper-V	Red Hat Customer Portal Certification Catalogue에 기재된 Hyper-V 버전에서 호스트됩니다.
VMware ESX 및 ESXi	Red Hat Customer Portal Certification Catalogue에 기재된 ESX 및 ESXi 버전에서 호스트됩니다.

중요

Red Hat OpenStack Platform director를 사용하려면 최신 버전의 Red Hat Enterprise Linux 7이 호스트 운영 체제로 설치되어 있어야 합니다. 즉, 가상화 플랫폼이 기본 Red Hat Enterprise Linux 버전도 지원해야 합니다.

가상 머신 요구 사항

가상 언더클라우드의 리소스 요구 사항은 베어 메탈 언더클라우드의 리소스 요구 사항과 유사합니다. 프로비저닝 시 네트워크 모델, 게스트 CPU 기능, 스토리지 백엔드, 스토리지 포멧 및 캐싱 모드와 같은 다양한 튜닝 옵션을 고려해야 합니다.

네트워크 고려 사항

가상화된 언더클라우드의 다음 네트워크 고려 사항에 유의하십시오.

전원 관리

언더클라우드 VM은 오버클라우드 노드의 전원 관리 장치에 액세스해야 합니다. 이는 노드를 등록할 때 **pm_addr** 매개변수에 설정된 IP 주소입니다.

프로비저닝 네트워크

프로비저닝(**ctlplane**) 네트워크에 사용되는 NIC에는 오버클라우드 베어 메탈 노드의 NIC에 DHCP 요청을 브로드캐스트 및 제공하는 기능이 필요합니다. VM의 NIC를 베어 메탈 NIC와 동일한 네트워크에 연결하는 브릿지를 생성하는 것이 좋습니다.

참고

흔히 발생하는 문제는 하이퍼바이저 기술이 언더클라우드가 알 수 없는 주소에서 트래픽을 전송하지 못하도록 차단하는 것입니다. - Red Hat Enterprise Virtualization을 사용하는 경우 이를 방지하려면 **anti-mac-spoofing**을 비활성화하십시오. - VMware ESX 또는 ESXi를 사용하는 경우 이를 방지하려면 위장 전송을 허용하십시오. 해당 설정을 적용한 후 director VM의 전원을 껐다 켜야 합니다. VM을 재부팅하는 것만으로는 부족합니다.

아키텍처 예

다음은 KVM 서버를 사용하는 기본 언더클라우드 가상화 아키텍처의 예입니다. 네트워크 및 리소스 요구 사항에 따라 구축할 수 있는 기반이 되는 것입니다.

KVM 호스트는 두 개의 Linux 브리지를 사용합니다.

br-ex(eth0)

- 언더클라우드에 대한 외부 액세스 제공
- 외부 네트워크의 DHCP 서버는 가상 NIC(eth0)를 사용하여 언더클라우드에 네트워크 구성을 할당합니다.
- 베어 메탈 서버의 전원 관리 인터페이스에 액세스할 수 있는 언더클라우드 액세스 제공

br-ctlplane(eth1)

- 베어 메탈 오버클라우드 노드와 동일한 네트워크에 연결
- 언더클라우드는 가상 NIC(eth1)를 통해 DHCP 및 PXE 부팅 요청을 수행합니다.
- 오버클라우드의 베어 메탈 서버가 이 네트워크를 통해 PXE를 통해 부팅

이러한 브리지를 만들고 구성하는 방법에 대한 자세한 내용은 Red Hat Enterprise Linux 7 네트워킹 가이드의 "네트워크 브리징 구성" 을 참조하십시오.

KVM 호스트에는 다음 패키지가 필요합니다.

```
$ yum install libvirt-client libvirt-daemon qemu-kvm libvirt-daemon-driver-qemu libvirt-daemon-kvm
virt-install bridge-utils rsync virt-viewer
```

다음 명령은 KVM 호스트에 언더클라우드 가상 머신을 생성하고 해당 브릿지에 연결하는 가상 NIC 두 개를 생성합니다.

```
$ virt-install --name undercloud --memory=16384 --vcpus=4 --location /var/lib/libvirt/images/rhel-
server-7.5-x86_64-dvd.iso --disk size=100 --network bridge=br-ex --network bridge=br-ctlplane --
graphics=vnc --hvm --os-variant=rhel7
```

그러면 **libvirt** 도메인이 시작됩니다. **virt-manager** 로 연결하고 설치 프로세스를 안내합니다. 또는 다음 옵션을 사용하여 Kickstart 파일을 포함하도록 자동 설치를 수행할 수 있습니다.

```
--initrd-inject=/root/ks.cfg --extra-args "ks=file:/ks.cfg"
```

설치가 완료되면 **root** 사용자로 인스턴스에 SSH로 연결하고 의 지침을 따릅니다. *4장. 언더클라우드 설치*

백업

가상화된 언더클라우드를 백업하려면 다음과 같은 여러 가지 솔루션이 있습니다.

- **옵션 1:** Director undercloud 가이드의 백업 및 복원에 있는 지침을 따릅니다.
- **옵션 2:** 언더클라우드를 종료하고 언더클라우드 가상 머신 스토리지의 백업 사본을 가져옵니다.
- **옵션 3:** 하이퍼바이저에서 실시간 또는 원자성 스냅샷을 지원하는 경우 언더클라우드 VM의 스냅샷을 작성합니다.

KVM 서버를 사용하는 경우 다음 절차를 사용하여 스냅샷을 만듭니다.

1. **qemu-guest-agent** 가 언더클라우드 게스트 VM에서 실행 중인지 확인합니다.
2. 실행 중인 VM의 실시간 스냅샷을 생성합니다.

```
$ virsh snapshot-create-as --domain undercloud --disk-only --atomic --quiesce
```

1. (현재 읽기 전용) QCOW 백업 파일을 복사합니다.

```
$ rsync --sparse -avh --progress /var/lib/libvirt/images/undercloud.qcow2 1.qcow2
```

1. QCOW 오버레이 파일을 백업 파일에 병합하고 원래 파일을 사용하여 언더클라우드 VM을 다시 전환합니다.

```
$ virsh blockcommit undercloud vda --active --verbose --pivot
```

2.3. 네트워킹 요구 사항

언더클라우드 호스트에는 두 개 이상의 네트워크가 필요합니다.

- 프로비저닝 네트워크 - 오버클라우드에서 사용할 베어 메탈 시스템을 검색하는 데 도움이 되는 DHCP 및 PXE 부팅 기능을 제공합니다. 일반적으로 이 네트워크는 director가 PXE 부팅 및 DHCP 요청을 제공하도록 트렁크된 인터페이스에서 기본 VLAN을 사용해야 합니다. 일부 서버 하드웨어 BIOS는 VLAN에서 PXE 부팅을 지원하지만 BIOS에서는 부팅 후 VLAN을 기본 VLAN으로 변환하도록 지원해야 합니다. 그렇지 않으면 언더클라우드에 연결할 수 없습니다. 현재 서버 하드웨어의

작은 서브 세트만 이 기능을 완전히 지원합니다. 또한 모든 오버클라우드 노드에서 IPMI(Intelligent Platform Management Interface)를 통해 전원 관리를 제어하는 데 사용하는 네트워크이기도 합니다.

- 외부 네트워크 - 오버클라우드 및 언더클라우드에 대한 외부 액세스를 위한 별도의 네트워크입니다. 이 네트워크에 연결하는 인터페이스에는 정적으로 정의되거나 외부 DHCP 서비스를 통해 동적으로 라우팅 가능한 IP 주소가 필요합니다.

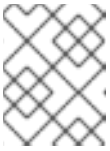
이는 필요한 최소 네트워크 수를 나타냅니다. 그러나 director는 다른 Red Hat OpenStack Platform 네트워크 트래픽을 다른 네트워크로 분리할 수 있습니다. Red Hat OpenStack Platform은 네트워크 격리를 위해 물리적 인터페이스와 태그된 VLAN을 모두 지원합니다.

다음을 확인합니다.

- 일반적인 최소 오버클라우드 네트워크 구성은 다음과 같습니다.
  - 단일 NIC 구성 - 기본 VLAN과 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN에서 프로비저닝 네트워크용 NIC 1개
  - 듀얼 NIC 구성 - 프로비저닝 네트워크용 NIC 1개 및 외부 네트워크용 다른 NIC
  - 듀얼 NIC 구성 - 기본 VLAN의 프로비저닝 네트워크용 NIC 1개 및 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN용 다른 NIC
  - 다중 NIC 구성 - 각 NIC에서 다른 오버클라우드 네트워크 유형에 서브넷 사용
- 추가 물리적 NIC는 개별 네트워크를 격리하거나 본딩된 인터페이스를 생성하거나 태그된 VLAN 트래픽을 위임하는 데 사용할 수 있습니다.
- VLAN을 사용하여 네트워크 트래픽 유형을 분리하는 경우 802.1Q 표준을 지원하는 스위치를 사용하여 태그된 VLAN을 제공합니다.
- 오버클라우드 생성 중에 모든 오버클라우드 머신에서 단일 이름을 사용하여 NIC를 참조합니다. 혼동하지 않도록 각 오버클라우드 노드에서 각각의 해당 네트워크에 대해 동일한 NIC를 사용하는 것이 좋습니다. 예를 들어 프로비저닝 네트워크에는 기본 NIC를 사용하고, OpenStack 서비스에는 보조 NIC를 사용합니다.
- 프로비저닝 네트워크 NIC가 director 머신의 원격 연결에 사용되는 NIC와 동일하지 않은지 확인합니다. director 설치는 프로비저닝 NIC를 사용하여 브릿지를 생성하여 원격 연결을 모두 해제합니다. director 시스템에 대한 원격 연결에는 외부 NIC를 사용합니다.
- 프로비저닝 네트워크에는 현재 환경 크기에 맞는 IP 범위가 필요합니다. 다음 지침에 따라 이 범위에 포함할 총 IP 주소 수를 결정하십시오.
  - 프로비저닝 네트워크에 연결된 노드당 IP 주소를 1개 이상 포함합니다.
  - 고가용성 구성을 계획하는 경우 클러스터의 가상 IP에 대한 추가 IP 주소를 포함합니다.
  - 환경 확장을 위해 범위 내에 추가 IP 주소를 포함합니다.

참고

프로비저닝 네트워크에서 중복 IP 주소를 피해야 합니다. 자세한 내용은 3.2절. "계획 네트워크"의 내용을 참조하십시오.



참고

스토리지, 공급자 및 테넌트 네트워크와 같이 IP 주소 사용량을 계획하는 방법에 대한 자세한 내용은 네트워킹 가이드를 참조하십시오.

- 프로비저닝 NIC를 PXE로 부팅하도록 모든 오버클라우드 시스템을 설정하고, 외부 NIC(및 시스템의 다른 모든 NIC)에서 PXE 부팅을 비활성화합니다. 또한 프로비저닝 NIC의 부팅 순서에서 PXE 부팅이 하드 디스크 및 CD/DVD 드라이브보다 우선하도록 맨 위 순서로 지정합니다.

- 모든 오버클라우드 베어 메탈 시스템에는 IPMI(Intelligent Platform Management Interface)와 같은 지원되는 전원 관리 인터페이스가 필요합니다. 이를 통해 director에서 각 노드의 전원 관리를 제어할 수 있습니다.

- 각 오버클라우드 시스템에 대해 프로비저닝 NIC의 MAC 주소, IPMI NIC의 IP 주소, IPMI 사용자 이름 및 IPMI 비밀번호를 기록해 두십시오. 이 정보는 나중에 오버클라우드 노드를 설정할 때 유용합니다.

- 외부 인터넷에서 인스턴스에 액세스할 필요가 있는 경우 공용 네트워크에서 유동 IP 주소를 할당하고 이 주소를 인스턴스와 연결할 수 있습니다. 인스턴스는 해당 개인 IP를 보유하고 있지만, 네트워크 트래픽에서 NAT를 사용하여 유동 IP 주소를 통과합니다. 유동 IP 주소는 여러 개인 IP 주소가 아니라 단일 인스턴스에만 할당할 수 있습니다. 하지만 유동 IP 주소는 단일 테넌트에서만 사용하도록 지정되어 있으므로 테넌트가 필요에 따라 특정 인스턴스와 연결하거나 연결을 해제할 수 있습니다. 이 설정을 사용하면 해당 인프라가 외부 인터넷에 노출되므로 적합한 보안 관행을 준수하고 있는지 확인해야 합니다.

- 지정된 브릿지의 멤버로 단일 인터페이스 또는 단일 본딩만 사용하면 Open vSwitch에서 네트워크 루프 발생 위험을 완화할 수 있습니다. 여러 개의 본딩이나 인터페이스가 필요한 경우 여러 브릿지를 설정할 수 있습니다.

- 오버클라우드 노드가 Red Hat Content Delivery Network 및 네트워크 시간 서버와 같은 외부 서비스에 연결할 수 있도록 DNS 호스트 이름 확인을 사용하는 것이 좋습니다.

- 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 방법을 참조하십시오.

중요

OpenStack Platform 구현의 보안 수준은 네트워크 환경의 보안 수준에 따라 좌우됩니다. 네트워킹 환경에서 적합한 보안 원칙에 따라 네트워크 액세스가 적절히 제어되는지 확인합니다. 예를 들면 다음과 같습니다.

- 네트워크 세그멘테이션을 사용하여 네트워크 이동을 줄이고 민감한 데이터를 분리합니다. 플랫 네트워크는 보안 수준이 훨씬 낮습니다.
- 서비스 액세스 및 포트를 최소로 제한합니다.
- 적절한 방화벽 규칙과 암호를 사용합니다.
- SELinux를 활성화합니다.

시스템 보안에 대한 자세한 내용은 다음을 참조하십시오.

- Red Hat Enterprise Linux 7 보안 가이드
- Red Hat Enterprise Linux 7 SELinux 사용자 및 관리자 가이드

2.4. 오버클라우드 요구 사항

다음 섹션에서는 오버클라우드 설치의 개별 시스템 및 노드에 대한 요구 사항에 대해 자세히 설명합니다.

2.4.1. 컴퓨팅 노드 요구 사항

컴퓨팅 노드는 가상 머신 인스턴스가 시작된 후 이를 실행하는 역할을 합니다. 컴퓨팅 노드는 하드웨어 가상화를 지원해야 합니다. 또한 호스팅하는 가상 머신 인스턴스의 요구 사항을 지원하기에 충분한 메모리 및 디스크 공간이 있어야 합니다.

프로세서

- Intel 64 또는 AMD64 CPU 확장 기능을 지원하고, AMD-V 또는 Intel VT 하드웨어 가상화 확장 기능이 활성화된 64비트 x86 프로세서. 이 프로세서에 최소 4개의 코어가 탑재되어 있는 것이 좋습니다.
- IBM POWER 8 프로세서

메모리

최소 6GB의 RAM. 가상 머신 인스턴스에서 사용하려는 메모리 크기에 따라 기본 메모리 요구 사항에 RAM을 추가합니다.

디스크 공간

최소 50GB의 사용 가능한 디스크 공간이 필요합니다.

네트워크 인터페이스 카드

최소 1개의 1Gbps 네트워크 인터페이스 카드가 필요합니다. 프로덕션 환경에서는 적어도 두 개 이상의 NIC를 사용하는 것이 좋습니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 네트워크 인터페이스 카드를 사용합니다.

전원 관리

각 컴퓨팅 노드에는 서버 마더보드의 IPMI(Intelligent Platform Management Interface) 기능과 같이 지원되는 전원 관리 인터페이스가 필요합니다.

2.4.2. 컨트롤러 노드 요구 사항

컨트롤러 노드는 Red Hat OpenStack Platform 환경에서 Horizon 대시보드, 백엔드 데이터베이스 서버, Keystone 인증, 고가용성 서비스와 같은 핵심 서비스를 호스팅합니다.

프로세서

Intel 64 또는 AMD64 CPU 확장을 지원하는 64비트 x86 프로세서입니다.

메모리

최소 메모리 용량은 32GB입니다. 하지만 권장 메모리 용량은 vCPU 수(CPU 코어 수와 하이퍼 스레딩 값을 곱한 값)에 따라 다릅니다. 다음 계산을 지침으로 사용합니다.

- **컨트롤러의 최소 RAM 계산:**
  - vCPU마다 1.5GB 메모리를 사용합니다. 예를 들어 48개의 vCPU가 있는 머신에는 72GB의 RAM이 있어야 합니다.
- **컨트롤러의 권장 RAM 계산:**
  - vCPU마다 3GB 메모리를 사용합니다. 예를 들어 48개의 vCPU가 있는 머신에는 144GB RAM이 있어야 합니다.

메모리 요구 사항 측정에 대한 자세한 내용은 Red Hat 고객 포털에서 "고가용성 컨트롤러에 대한 Red Hat OpenStack Platform 하드웨어 요구 사항" 을 참조하십시오.

디스크 스토리지 및 레이아웃

컨트롤러 노드에서 Object Storage 서비스(swift)를 실행하지 않는 경우 최소 50GB의 스토리지가 필요합니다. Telemetry(**gnocchi**) 및 Object Storage 서비스는 모두 컨트롤러에 설치되고 root 디스크를 사용하도록 구성됩니다. 이러한 기본값은 상용 하드웨어에 구축된 소형 오버클라우드 배포에 적합합니다. 이러한 환경은 일반적인 개념 증명 및 테스트 환경입니다. 이러한 기본값을 사용하면 워크로드 용량 및 성능 면에서는 떨어지지만 최소의 플래닝으로 오버클라우드 배포가 가능합니다.
하지만 엔터프라이즈 환경에서는 Telemetry가 스토리지에 지속적으로 액세스하므로 이 경우 심각한 성능 장애가 발생할 수 있습니다. 그러면 디스크 I/O 사용이 과해지고 다른 모든 Controller 서비스의 성능에 심각한 영향을 미칩니다. 이러한 유형의 환경에서는 오버클라우드를 계획하고 적절하게 설정해야 합니다.

Red Hat은 Telemetry와 Object Storage에 대한 여러 가지 설정 권장 사항을 제공합니다. 자세한 내용은 Deployment Recommendations for Specific Red Hat OpenStack Platform Services 를 참조하십시오.

네트워크 인터페이스 카드

최소 2개의 1GB의 네트워크 인터페이스 카드가 필요합니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 네트워크 인터페이스 카드를 사용합니다.

전원 관리

각 컨트롤러 노드에는 서버의 마더보드에서 IPMI(Intelligent Platform Management Interface) 기능과 같은 지원되는 전원 관리 인터페이스가 필요합니다.

2.4.2.1. 가상화 지원

Red Hat은 Red Hat Virtualization 플랫폼에서 가상화된 컨트롤러 노드만 지원합니다. 자세한 내용은 가상화된 컨트롤 플레인을 참조하십시오.

2.4.3. Ceph Storage 노드 요구 사항

Ceph Storage 노드는 Red Hat OpenStack Platform 환경에 스토리지 오브젝트를 제공합니다.

배치 그룹

Ceph는 배치 그룹을 사용하여 대규모의 효율적인 동적 오브젝트 추적을 지원합니다. OSD 오류가 발생하거나 클러스터를 재조정하는 경우 Ceph에서 배치 그룹 및 해당 콘텐츠를 이동하거나 복제할 수 있으므로, Ceph 클러스터의 효율적인 재조정 및 복구가 가능합니다. Director가 생성하는 기본 배치 그룹 수가 항상 최적 개수인 것은 아니므로 요구 사항에 따라 올바른 배치 그룹 수를 계산하는 것이 중요합니다. 배치 그룹 계산기를 사용하여 올바른 개수를 계산할 수 있습니다. 풀당 Ceph PG(배치 그룹) 계산기

프로세서

Intel 64 또는 AMD64 CPU 확장을 지원하는 64비트 x86 프로세서입니다.

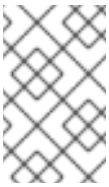
메모리

일반적으로 OSD 호스트당 16GB의 RAM과 OSD 데몬당 추가 2GB의 RAM으로 기준선을 정하는 것이 좋습니다.

디스크 레이아웃

크기 조정은 필요한 스토리지에 따라 다릅니다. Red Hat Ceph Storage 노드의 권장 설정에는 다음과 같은 레이아웃의 디스크가 3개 이상 필요합니다.

- **/dev/sda** - root 디스크입니다. director가 주요 오버클라우드 이미지를 디스크에 복사합니다. 최소 50GB의 사용 가능한 디스크 공간이 있어야 합니다.
- **/dev/sdb** - 저널 디스크입니다. 이 디스크는 Ceph OSD 저널용 파티션으로 나뉩니다. 예를 들면 **/dev/sdb1**,**/dev/sdb2**,**/dev/sdb3** 등이 있습니다. 저널 디스크는 일반적으로 시스템 성능을 지원하기 위한 SSD(솔리드 스테이트 드라이브)입니다.
- **/dev/sdc** 이후 - OSD 디스크입니다. 스토리지 요구 사항에 따라 필요한 개수만큼 디스크를 사용합니다.



참고

Red Hat OpenStack Platform director는 **ceph-ansible**을 사용하지만 Ceph Storage 노드의 root 디스크에 OSD 설치를 지원하지 않습니다. 즉, 지원되는 Ceph Storage 노드에 대해 두 개 이상의 디스크가 필요합니다.

네트워크 인터페이스 카드

최소 1개의 1Gbps 네트워크 인터페이스 카드가 필요합니다. 프로덕션 환경에서는 적어도 두 개 이상의 NIC를 사용하는 것이 좋습니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 네트워크 인터페이스 카드를 사용하십시오. 대량의 트래픽에 서비스를 제공하는 OpenStack Platform 환경을 구축하는 경우 스토리지 노드에 10Gbps 인터페이스를 사용하는 것이 좋습니다.

전원 관리

각 컨트롤러 노드에는 서버의 마더보드에서 IPMI(Intelligent Platform Management Interface) 기능과 같은 지원되는 전원 관리 인터페이스가 필요합니다.

이미지 속성

Red Hat Ceph Storage 블록 장치 성능을 개선하기 위해 **virtio-scsi** 드라이버를 사용하도록 Glance 이미지를 구성할 수 있습니다. 이미지에 권장되는 이미지 속성에 대한 자세한 내용은 *Red Hat Ceph Storage* 설명서에서 Glance Congfiguring Glance 를 참조하십시오.

Ceph Storage 클러스터를 사용한 오버클라우드 설치에 대한 자세한 내용은 Deploying an Overcloud with Containerized Red Hat Ceph 가이드를 참조하십시오.

2.4.4. 오브젝트 스토리지 노드 요구 사항

Object Storage 노드는 오버클라우드에 오브젝트 스토리지 계층을 제공합니다. Object Storage 프록시는 컨트롤러 노드에 설치됩니다. 스토리지 계층에는 노드당 여러 개의 디스크가 있는 베어 메탈 노드가 필요합니다.

프로세서

Intel 64 또는 AMD64 CPU 확장을 지원하는 64비트 x86 프로세서입니다.

메모리

메모리 요구 사항은 스토리지 공간의 크기에 따라 다릅니다. 가장 좋은 방법은 1TB의 하드 디스크 공간마다 최소 1GB의 메모리를 사용하는 것입니다. 최적의 성능을 위해 특히 워크로드가 작은 파일(100GB가 없는 경우) 1TB의 하드 디스크 공간당 2GB를 사용하는 것이 좋습니다.

디스크 공간

스토리지 요구 사항은 워크로드에 필요한 용량에 따라 다릅니다. 계정 및 컨테이너 데이터를 저장하기 위해서는 SSD 드라이브를 사용하는 것이 좋습니다. 오브젝트에 대한 계정과 컨테이너 데이터의 용량 비율은 약 1%입니다. 예를 들어 100TB의 모든 하드 드라이브 용량마다 계정과 컨테이너 데이터에 1TB의 SSD 용량을 준비하도록 합니다.
하지만 이는 저장된 데이터 유형에 따라 달라집니다. 주로 작은 오브젝트를 구성하는 경우 더 많은 SSD 공간을 제공합니다. 큰 오브젝트(비디오, 백업)의 경우에는 SSD의 용량을 줄일 수 있습니다.

디스크 레이아웃

권장 노드 구성에는 다음과 유사한 디스크 레이아웃이 필요합니다.

- **/dev/sda** - root 디스크입니다. director가 주요 오버클라우드 이미지를 디스크에 복사합니다.
- **/dev/sdb** - 계정 데이터에 사용됩니다.
- **/dev/sdc** - 컨테이너 데이터에 사용됩니다.
- **/dev/sdd** 이후 - 오브젝트 서버 디스크입니다. 스토리지 요구 사항에 따라 필요한 개수만큼 디스크를 사용합니다.

네트워크 인터페이스 카드

최소 2개의 1GB의 네트워크 인터페이스 카드가 필요합니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 네트워크 인터페이스 카드를 사용합니다.

전원 관리

각 컨트롤러 노드에는 서버의 마더보드에서 IPMI(Intelligent Platform Management Interface) 기능과 같은 지원되는 전원 관리 인터페이스가 필요합니다.

2.5. 리포지토리 요구 사항

언더클라우드와 오버클라우드 모두 CDN(Red Hat Content Delivery Network) 또는 Red Hat Satellite Server 5 또는 Red Hat Satellite Server 6을 통해 Red Hat 리포지토리에 액세스해야 합니다. Red Hat Satellite Server를 사용하려면 필요한 리포지토리를 OpenStack Platform 환경에 동기화해야 합니다. 다음 CDN 채널 이름 목록을 가이드로 사용합니다.

표 2.1. OpenStack Platform 리포지토리

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 7 Server(RPM)	**rhel-7-server-rpms**	x86_64 시스템용 기본 운영 체제 리포지토리입니다.

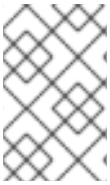
이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 7 Server - Extras(RPM)	**rhel-7-server-extras-rpms**	Red Hat OpenStack Platform 종속성을 포함합니다.
Red Hat Enterprise Linux 7 Server - RH Common(RPM)	**rhel-7-server-rh-common-rpms**	Red Hat OpenStack Platform 배포 및 구성을 위한 툴을 포함합니다.
Red Hat Satellite Tools 6.3 (for RHEL 7 Server)(RPM) x86_64	**rhel-7-server-satellite-tools-6.3-rpms**	Red Hat Satellite Server 6으로 호스트를 관리하는 툴입니다. 이후 버전의 Satellite Tools 리포지토리를 사용하면 언더클라우드 설치에 실패할 수 있습니다.
Red Hat Enterprise Linux High Availability(for RHEL 7 Server) (RPM)	**rhel-ha-for-rhel-7-server-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다. 컨트롤러 노드 고가용성에 사용됩니다.
Red Hat OpenStack Platform 13 for RHEL 7(RPM)	**rhel-7-server-openstack-13-rpms**	코어 Red Hat OpenStack Platform 리포지토리입니다. Red Hat OpenStack Platform director의 패키지도 포함합니다.
Red Hat Ceph Storage OSD 3 for Red Hat Enterprise Linux 7 Server(RPM)	**rhel-7-server-rhceph-3-osd-rpms**	(Ceph Storage 노드용) Ceph Storage Object Storage 데몬용 리포지토리입니다. Ceph Storage 노드에 설치됩니다.
Red Hat Ceph Storage MON 3 for Red Hat Enterprise Linux 7 Server(RPM)	**rhel-7-server-rhceph-3-mon-rpms**	(Ceph Storage 노드용) Ceph Storage 모니터 데몬용 리포지토리입니다. Ceph Storage 노드를 사용하는 OpenStack 환경의 Controller 노드에 설치됩니다.
Red Hat Ceph Storage Tools 3 for Red Hat Enterprise Linux 7 Server(RPM)	**rhel-7-server-rhceph-3-tools-rpms**	노드가 Ceph Storage 클러스터와 통신할 수 있는 툴을 제공합니다. Ceph Storage 클러스터와 함께 오버클라우드를 배포하거나 오버클라우드를 기존 Ceph Storage 클러스터와 통합할 때 모든 노드에 대해 이 리포지토리를 활성화합니다.
Red Hat OpenStack 13 Director Deployment Tools for RHEL 7(RPM)	**rhel-7-server-openstack-13-deployment-tools-rpms**	(Ceph Storage 노드용) 현재 Red Hat OpenStack Platform director 버전과 호환되는 배포 툴 세트를 제공합니다. 활성 Red Hat OpenStack Platform 서브스크립션이 없는 Ceph 노드에 설치됩니다.

이름	리포지토리	요구 사항 설명
Enterprise Linux for Real Time for NFV(RHEL 7 Server)(RPM)	**rhel-7-server-nfv-rpms**	NFV용 실시간 KVM(RT-KVM) 리포지토리로, 실시간 커널을 활성화하는 패키지가 포함되어 있습니다. 이 리포지토리는 RT-KVM을 대상으로 하는 모든 컴퓨팅 노드에 대해 활성화되어야 합니다. 알림: 이 리포지토리에 액세스하려면 별도의 **Red Hat OpenStack Platform for Real Time** SKU 서브스크립션이 필요합니다.
Red Hat OpenStack Platform 13 ELS (Extended Life Cycle Support for RHEL 7)	**rhel-7-server-openstack-13-els-rpms**	2021년 6월 26일에 시작된 연장된 라이브 사이클 지원에 대한 업데이트가 포함되어 있습니다. 이 리포지토리를 사용하려면 "OpenStack 13 플랫폼 ELS (Extended Life Cycle Support)" (MCT3637)가 필요합니다.

IBM POWER용 OpenStack Platform 리포지토리

이러한 리포지토리는 부록 *G. Red Hat OpenStack Platform for POWER* 기능에서 사용됩니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux for IBM Power, little endian	**rhel-7-for-power-le-rpms**	ppc64le 시스템용 기본 운영 체제 리포지토리입니다.
Red Hat OpenStack Platform 13 for RHEL 7(RPM)	**rhel-7-server-openstack-13-for-power-le-rpms**	ppc64le 시스템용 Red Hat OpenStack Platform 코어 리포지토리입니다.



참고

오프라인 네트워크에서 Red Hat OpenStack Platform 환경에 대한 리포지토리를 구성하려면 Red Hat Customer Portal의 "오프라인 환경에서 Red Hat OpenStack Platform Director 구성" 을 참조하십시오.

3장. 오버클라우드 계획

다음 섹션에서는 Red Hat OpenStack Platform 환경의 다양한 측면을 계획하기위한 몇 가지 지침을 제공합니다. 여기에는 노드 역할 정의, 네트워크 토폴로지 계획 및 스토리지가 포함됩니다.

중요

배포 후 오버클라우드 노드의 이름을 바꾸지 마십시오. 배포 후 노드 이름을 변경하면 인스턴스 관리 문제가 발생합니다.

3.1. 노드 배포 역할 계획

director는 오버클라우드 빌드에 필요한 여러 개의 기본 노드 유형을 제공합니다. 이러한 노드 유형은 다음과 같습니다.

컨트롤러

환경을 제어하기 위한 주요 서비스를 제공합니다. 여기에는 대시보드(horizon), 인증(keystone), 이미지 스토리지(glance), 네트워킹(neutron), 오케스트레이션(heat) 및 고가용성 서비스가 포함됩니다. Red Hat OpenStack Platform 환경에는 프로덕션 수준의 고가용성 환경을 위한 컨트롤러 노드 3개가 필요합니다.

참고

하나의 노드로 구성된 환경은 프로덕션이 아닌 테스트 목적으로만 사용할 수 있습니다. 두 개의 노드 또는 세 개 이상의 노드로 구성된 환경은 지원되지 않습니다.

Compute

하이퍼바이저 역할을 하고 환경에서 가상 머신을 실행하는 데 필요한 처리 기능을 제공하는 물리 서버입니다. 기본 Red Hat OpenStack Platform 환경에는 컴퓨팅 노드가 적어도 한 개 이상 필요합니다.

Ceph Storage

Red Hat Ceph Storage를 제공하는 호스트입니다. 추가 Ceph Storage 호스트는 클러스터에서 확장될 수 있습니다. 이 배포 역할은 선택 사항입니다.

Swift Storage

OpenStack의 swift 서비스에 외부 오브젝트 스토리지를 제공하는 호스트입니다. 이 배포 역할은 선택 사항입니다.

다음 표에서는 다른 오버클라우드의 몇 가지 예제를 보여주고 각 시나리오에 사용되는 노드 유형을 정의합니다.

표 3.1. 시나리오에 사용되는 노드 배포 역할

	컨트롤러	컴퓨팅	Ceph Storage	Swift Storage	합계
소규모 오버클라우드	3	1	-	-	4
중간 규모 오버클라우드	3	3	-	-	6

추가 오브젝트 스토리지가 있는 중간 규모의 오버클라우드	3	3	-	3	9
Ceph Storage 클러스터가 있는 중간 규모의 오버클라우드	3	3	3	-	9

또한 개별 서비스를 사용자 지정 역할로 나눌지 여부를 검토합니다. 구성 가능한 역할 아키텍처에 대한 자세한 내용은 *Advanced Overcloud Customization* 가이드의 "Composable Services and Custom Roles" 를 참조하십시오.

3.2. 계획 네트워크

역할 및 서비스를 적절하게 매핑하여 서로 올바르게 통신할 수 있도록 환경의 네트워킹 토폴로지와 서브넷을 계획하는 것이 중요합니다. Red Hat OpenStack Platform은 자율적으로 작동하고 소프트웨어 기반 네트워크, 고정 및 유동 IP 주소, DHCP를 관리하는 neutron 네트워킹 서비스를 사용합니다. director는 오버클라우드 환경의 각 Controller 노드에 이 서비스를 배포합니다.

Red Hat OpenStack Platform은 환경의 다양한 서브넷에 할당된 별도의 네트워크 트래픽 유형에 다양한 서비스를 매핑합니다. 이러한 네트워크 트래픽 유형에는 다음이 포함됩니다.

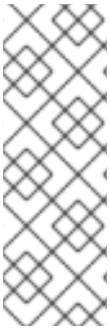
표 3.2. 네트워크 유형 할당

네트워크 유형	설명	사용 방법
IPMI	노드의 전원 관리에 사용되는 네트워크입니다. 이 네트워크는 언더클라우드를 설치하기 전에 미리 정의되어 있습니다.	모든 노드
프로비저닝/컨트롤 플레인	director는 이 네트워크 트래픽 유형을 사용하여 PXE 부팅을 통해 새 노드를 배포하고 오버클라우드 베어 메탈 서버에 OpenStack Platform 설치를 오케스트레이션합니다. 이 네트워크는 언더클라우드를 설치하기 전에 미리 정의되어 있습니다.	모든 노드
내부 API	내부 API 네트워크는 API 통신, RPC 메시지 및 데이터베이스 통신을 사용하여 OpenStack 서비스 간 통신에 사용됩니다.	Controller, Compute, Cinder Storage, Swift Storage

tenant	Neutron은 VLAN 분리(각 테넌트 네트워크가 네트워크 VLAN임) 또는 터널링(VXLAN 또는 GRE를 통해)을 사용하여 각 테넌트에 자체 네트워크를 제공합니다. 네트워크 트래픽은 각 테넌트 네트워크 내에서 격리됩니다. 각 테넌트 네트워크에는 IP 서브넷이 연결되어 있으며, 네트워크 네임스페이스는 여러 테넌트 네트워크가 충돌을 초래하지 않고 동일한 주소 범위를 사용할 수 있음을 의미합니다.	컨트롤러, Compute
스토리지	블록 스토리지, NFS, iSCSI 등. 이 방법은 성능상의 이유로 완전히 별도의 스위치 패브릭에 격리됩니다.	모든 노드
스토리지 관리	OpenStack Object Storage(swift)는 이 네트워크를 사용하여 참여하는 복제본 노드 간에 데이터 오브젝트를 동기화합니다. 프록시 서비스는 사용자 요청과 기본 스토리지 계층 간의 중간 인터페이스 역할을 합니다. 프록시는 들어오는 요청을 수신하고 요청된 데이터를 검색하는 데 필요한 복제본을 찾습니다. Ceph 백엔드를 사용하는 서비스는 Ceph와 직접 상호 작용하지 않고 프런트 엔드 서비스를 사용하므로 스토리지 관리 네트워크를 통해 연결됩니다. 이 트래픽이 Ceph에 직접 연결되므로 RBD 드라이버는 예외입니다.	Controller, Ceph Storage, Cinder Storage, Swift Storage
스토리지 NFS	이 네트워크는 CephFS를 NFS 백엔드에 매핑하기 위해 **ganesha** 서비스와 함께 Shared File System 서비스(manila)를 사용하는 경우에만 필요합니다.	컨트롤러
외부	그래픽 시스템 관리용 OpenStack 대시보드(horizon), OpenStack 서비스의 공용 API를 호스팅하고, 인스턴스로 향하는 들어오는 트래픽에 대해 SNAT를 수행합니다. 외부 네트워크에서 개인 IP 주소를 사용하는 경우(RFC-1918에 따라), 인터넷에서 시작되는 트래픽에 대해 추가 NAT를 수행해야 합니다.	컨트롤러 및 언더클라우드

유동 IP	유동 IP 주소 간 1-to-1 IP 주소 매핑과 테넌트 네트워크의 인스턴스에 실제로 할당된 IP 주소를 사용하여 들어오는 트래픽이 인스턴스에 도달할 수 있습니다. 외부와 별도로 VLAN에서 유동 IP를 호스팅하는 경우 유동 IP VLAN을 컨트롤러 노드로 트렁크하고 오버클라우드 생성 후 Neutron을 통해 VLAN을 추가할 수 있습니다. 이를 통해 여러 브리지에 연결된 여러 유동 IP 네트워크를 만들 수 있습니다. VLAN은 트렁크되지만 인터페이스로 구성되지 않습니다. 대신 neutron은 각 유동 IP 네트워크에 대해 선택한 브릿지에 VLAN 세그먼트 ID를 사용하여 OVS 포트를 생성합니다.	컨트롤러
관리	SSH 액세스, DNS 트래픽 및 NTP 트래픽과 같은 시스템 관리 기능에 대한 액세스를 제공합니다. 이 네트워크도 비Controller 노드에 대한 게이트웨이 역할을 합니다.	모든 노드

일반적인 Red Hat OpenStack Platform 설치에서는 종종 네트워크 유형의 수가 물리적 네트워크 연결 수를 초과합니다. 모든 네트워크를 적절한 호스트에 연결하기 위해 오버클라우드에서는 VLAN 태그를 사용하여 하나의 인터페이스로 두 개 이상의 네트워크를 제공합니다. 대부분의 네트워크는 서브넷이 분리되어 있지만, 일부는 인터넷 액세스 또는 인프라 네트워크 연결에 라우팅을 제공하기 위해 계층 3 게이트웨이가 필요합니다.



참고

배포 시 neutron VLAN 모드(터널링이 비활성화된 상태)를 사용하려는 경우에도 프로젝트 네트워크(GRE 또는 VXLAN으로 터널링됨)를 배포하는 것이 좋습니다. 이 경우 배포 시 약간의 사용자 정의 작업이 필요하며, 이후에 유틸리티 네트워크 또는 가상화 네트워크로 터널 네트워크를 사용할 수 있는 옵션은 그대로 유지됩니다. VLAN을 사용하여 테넌트 네트워크를 만들지만, 테넌트 VLAN을 사용하지 않고 네트워크를 특별한 용도로 사용하기 위한 VXLAN 터널을 만들 수도 있습니다. 테넌트 VLAN을 사용한 배포에 VXLAN 기능을 추가할 수 있지만, 기존 오버클라우드에 테넌트 VLAN을 추가하려면 서비스를 중단해야 합니다.

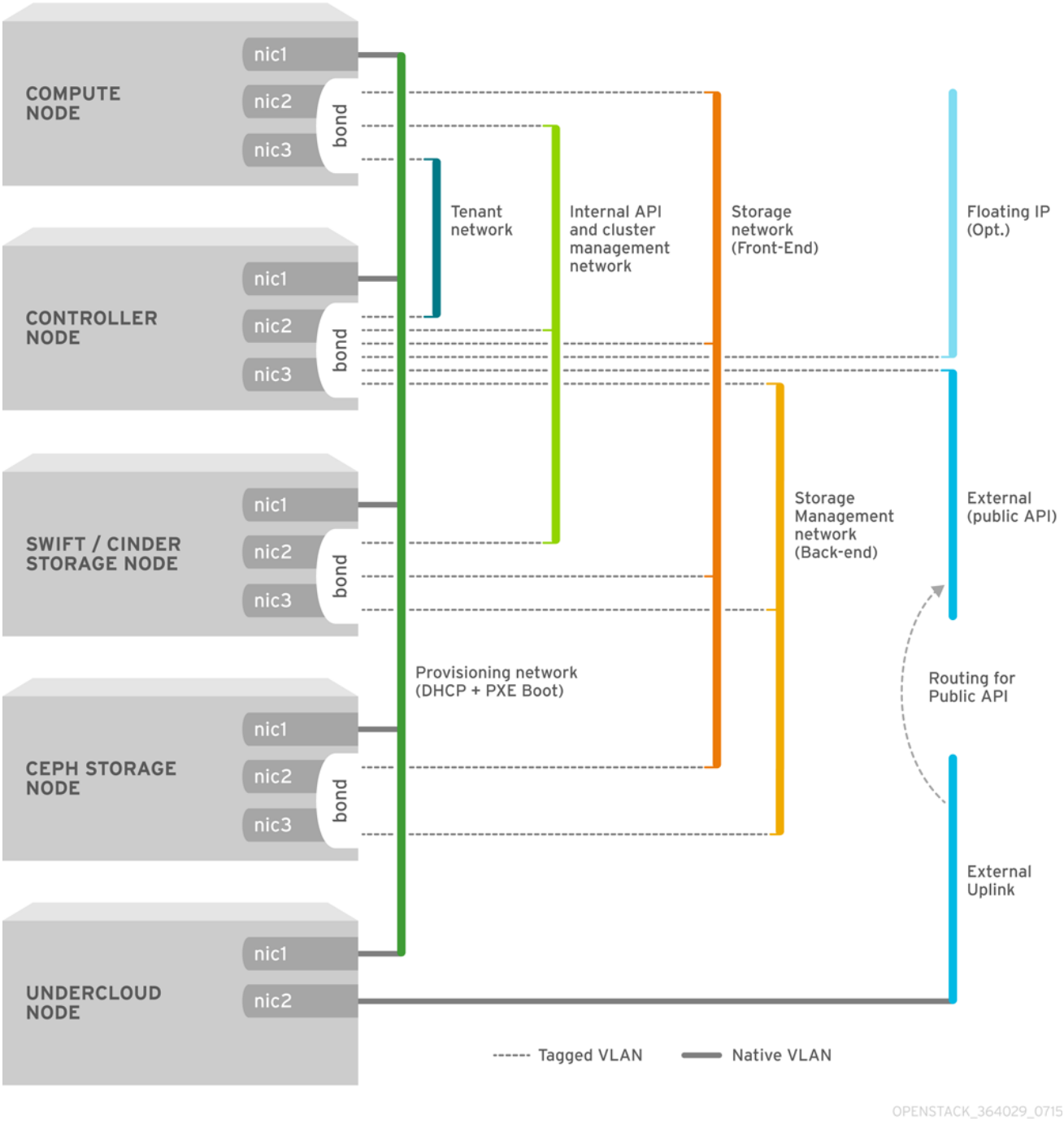
director는 이러한 트래픽 유형 6개를 특정 서브넷 또는 VLAN에 매핑하는 방법을 제공합니다. 이러한 트래픽 유형은 다음과 같습니다.

- 내부 API
- 스토리지
- 스토리지 관리
- 테넌트 네트워크
- 외부
- 관리 (선택 사항)

할당되지 않은 모든 네트워크는 프로비저닝 네트워크와 동일한 서브넷에 자동으로 할당됩니다.

아래 다이어그램은 네트워크가 별도의 VLAN에서 분리된 네트워크 토폴로지의 예를 제공합니다. 각 오버클라우드 노드는 본딩에 두 개의 인터페이스(**nic2 및 nic3)**를 사용하여 해당 VLAN에 이러한 네트워크를 제공합니다. 각 오버클라우드 노드는 **nic1** 을 사용하여 기본 VLAN을 통해 프로비저닝 네트워크를 통해 언더클라우드와 통신합니다.

그림 3.1. 연결된 인터페이스를 사용하는 VLAN 토폴로지의 예.

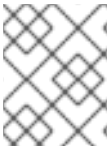


다음 표에서는 다양한 네트워크 레이아웃을 매핑하는 네트워크 트래픽의 예를 보여줍니다.

표 3.3. 네트워크 매핑

	매핑	총 인터페이스	총 VLAN

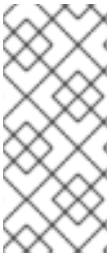
외부 액세스가 가능한 플랫 네트워크	네트워크 1 - 프로비저닝, 내부 API, 스토리지, 스토리지 관리, 테넌트 네트워크 네트워크 2 - 외부, 유동 IP(프로덕션 생성 후 매핑)	2	2
분리된 네트워크	네트워크 1 - 프로비저닝 네트워크 2 - 내부 API 네트워크 3 - 테넌트 네트워크 네트워크 4 - 스토리지 네트워크 5 - 스토리지 관리 Network 6 - Management (선택 사항) Network 7 - 외부, 유동 IP(프로덕션 생성 후 매핑)	3 (두 개의 결합된 인터페이스 포함)	7



참고

RHV(Red Hat Virtualization)를 사용하는 경우 오버클라우드 컨트롤 플레인을 가상화할 수 있습니다. 자세한 내용은 가상화된 컨트롤 플레인 생성 을 참조하십시오.

3.3. 계획 스토리지



참고

모든 드라이버 또는 백엔드 유형의 백엔드 cinder-volume을 사용하는 게스트 인스턴스에서 LVM을 사용하면 성능, 볼륨 가시성 및 가용성, 데이터 손상 관련 문제가 발생합니다. 이러한 문제는 LVM 필터를 사용하여 완화할 수 있습니다. 자세한 내용은 *Storage Guide* 의 섹션 2.1 Back Ends 및 KCS 문서 3213311, "Using LVM on a cinder volume exposes the data to the compute host"를 참조하십시오.

director는 오버클라우드 환경에 다양한 스토리지 옵션을 제공합니다. 여기에는 다음이 포함됩니다.

Ceph Storage 노드

director가 Red Hat Ceph Storage를 사용하여 확장 가능한 스토리지 노드 집합을 생성합니다. 오버클라우드는 이러한 노드를 사용하여 다음을 수행합니다.

- **이미지** - Glance는 VM의 이미지를 관리합니다. 이미지는 변경할 수 없습니다. OpenStack에서는 이미지를 바이너리 Blob으로 처리하고 그에 따라 이미지를 다운로드합니다. glance를 사용하여 이미지를 Ceph 블록 장치에 저장할 수 있습니다.
- **볼륨** - Cinder 볼륨은 블록 장치입니다. OpenStack에서는 볼륨을 사용하여 VM을 부팅하거나, 실행 중인 VM에 볼륨을 연결합니다. OpenStack에서는 Cinder 서비스를 사용하여 볼륨을 관리합니다. Cinder를 사용하면 이미지의 CoW(copy-on-write) 복제본을 사용하여 VM을 부팅할 수 있습니다.
- **파일 시스템** - Manila 공유는 파일 시스템에서 지원합니다. OpenStack 사용자는 Manila 서비스를 사용하여 공유를 관리합니다. Manila를 사용하면 Ceph Storage 노드의 데이터로 CephFS 파일 시스템에서 지원하는 공유를 관리할 수 있습니다.
- **게스트 디스크** - 게스트 디스크는 게스트 운영 체제 디스크입니다. 기본적으로 nova로 가상 머신을 부팅하면 해당 디스크는 하이퍼바이저의 파일 시스템에 파일로 표시됩니다(일반적으로 **/var/lib/nova/instances/<uuid>/**). Cinder를 사용하지 않고 Ceph 내부의 모든 가상 머신을 부팅할 수 있으므로 실시간 마이그레이션 프로세스를 통해 유지 관리 작업을 쉽게 수행할 수 있습니다. 또한 하이퍼바이저가 중지된 경우 **nova evacuate**를 트리거하고 가상 머신을 다른 위치에서 실행할 수 있으므로 편리합니다.

중요

지원되는 이미지 형식에 대한 자세한 내용은 *Instances and Images Guide* 의 Image Service 장을 참조하십시오.

자세한 내용은 Red Hat Ceph Storage Architecture Guide 를 참조하십시오.

Swift Storage 노드

director에서 외부 오브젝트 스토리지 노드를 생성합니다. 이는 오버클라우드 환경의 컨트롤러 노드를 확장하거나 교체해야 하지만 고가용성 클러스터 외부에서 오브젝트 스토리지를 유지해야 하는 경우에 유용합니다.

3.4. 계획 고가용성

고가용성 오버클라우드를 배포하기 위해 director는 여러 개의 컨트롤러 노드, 컴퓨팅 노드, 스토리지 노드가 단일 클러스터로 작동하도록 설정합니다. 노드 오류가 발생할 경우 해당 노드의 유형에 따라 자동 펜싱 및 다시 시작 프로세스가 트리거됩니다. 오버클라우드 고가용성 아키텍처 및 서비스에 관한 자세한 내용은 High Availability Deployment and Usage 를 참조하십시오.

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.

director(인스턴스 HA)를 사용하여 컴퓨팅 인스턴스의 고가용성을 설정할 수도 있습니다. 이 메커니즘은 노드에 오류가 발생할 경우 컴퓨팅 노드의 인스턴스를 자동으로 비우고 다시 생성합니다. 인스턴스 HA에 대한 요구 사항은 일반 오버클라우드 요구 사항과 동일하지만 몇 가지 추가 단계를 수행하여 배포할 환경을 준비해야 합니다. 인스턴스 HA가 작동하는 방법 및 설치 지침에 대한 자세한 내용은 High Availability for Compute Instances 가이드를 참조하십시오.

4장. 언더클라우드 설치

Red Hat OpenStack Platform 환경을 생성하는 첫 번째 단계는 언더클라우드 시스템에 director를 설치하는 것입니다. 여기에는 필요한 서브스크립션 및 리포지토리를 활성화하기 위한 몇 가지 사전 요구 사항이 포함됩니다.

4.1. 프록시를 사용하여 언더클라우드를 실행할 때 고려 사항

프록시를 사용하는 경우 다음 고려 사항을 검토하여 Red Hat OpenStack Platform의 여러 부분을 프록시와 통합하는 다양한 구성 방법과 각 방법의 제한 사항을 가장 잘 이해할 수 있습니다.

시스템 전체 프록시 구성

이 방법을 사용하여 언더클라우드의 모든 네트워크 트래픽에 대한 프록시 통신을 구성합니다. 프록시 설정을 구성하려면 **/etc/environment** 파일을 편집하고 다음 환경 변수를 설정합니다.

http_proxy

표준 HTTP 요청에 사용할 프록시입니다.

https_proxy

표준 HTTPs 요청에 사용할 프록시입니다.

no_proxy

프록시 통신에서 제외할, 쉼표로 구분된 도메인 목록입니다.

시스템 전체 프록시 방식에는 다음과 같은 제한 사항이 있습니다.

- **no_proxy** 변수는 주로 도메인 이름 (**www.example.com**), 도메인 접미사(**example.com**), 와일드카드가 있는 도메인(***.example.com**)을 주로 사용합니다. 대부분의 Red Hat OpenStack Platform 서비스는 **no_proxy** 의 IP 주소를 해석하지만 컨테이너 상태 점검과 같은 특정 서비스는 cURL 및 **wget** 의 제한으로 인해 **no_proxy** 환경 변수의 IP 주소를 해석하지 않습니다. 언더클라우드와 함께 시스템 전체 프록시를 사용하려면 설치 중에 **undercloud.conf** 파일에서 **container_healthcheck_disabled** 매개변수를 사용하여 컨테이너 상태 점검을 비활성화합니다. 자세한 내용은 BZ#1837458 - 컨테이너 상태 점검이 no_proxy CIDR 표기법을 준수 하지 않음을 참조하십시오.

- 일부 컨테이너가 **/etc/environments**의 환경 변수를 잘못 바인딩하고 구문 분석하므로 이러한 서비스를 실행할 때 문제가 발생합니다. 자세한 내용은 BZ#1916070 - /etc/environment 파일의 프록시 구성 업데이트가 컨테이너에서 올바르게 선택되지 않음 및 BZ#1918408 - mistral_executor 컨테이너가 no_proxy 환경 매개변수를 올바르게 설정하지 못함을 참조하십시오.

dnf 프록시 구성

모든 트래픽을 프록시를 통해 실행하도록 **dnf**를 구성하려면 이 방법을 사용합니다. 프록시 설정을 구성하려면 **/etc/dnf/dnf.conf** 파일을 편집하고 다음 매개변수를 설정합니다.

proxy

프록시 서버의 URL입니다.

proxy_username

프록시 서버에 연결하는 데 사용할 사용자 이름입니다.

proxy_password

프록시 서버에 연결하는 데 사용할 암호입니다.

proxy_auth_method

프록시 서버에서 사용하는 인증 방법입니다.

이러한 옵션에 대해 자세히 알아보려면 **man dnf.conf**를 실행합니다.

dnf 프록시 방식에는 다음과 같은 제한 사항이 있습니다.

- 이 방법은 **dnf**에 대해서만 프록시 지원을 제공합니다.
- **dnf** 프록시 방식에는 프록시 통신에서 특정 호스트를 제외하는 옵션이 포함되지 않습니다.

Red Hat Subscription Manager 프록시

이 방법을 사용하여 프록시를 통해 모든 트래픽을 실행하도록 Red Hat Subscription Manager를 구성합니다. 프록시 설정을 구성하려면 **/etc/rhsm/rhsm.conf** 파일을 편집하고 다음 매개변수를 설정합니다.

proxy_hostname

프록시에 대한 호스트입니다.

proxy_scheme

리포지토리 정의에 프록시를 작성할 때 프록시의 스키마입니다.

proxy_port

프록시의 포트입니다.

proxy_username

프록시 서버에 연결하는 데 사용할 사용자 이름입니다.

proxy_password

프록시 서버 연결에 사용할 암호입니다.

no_proxy

프록시 통신에서 제외하려는 특정 호스트의 호스트 이름 접미사 목록입니다(쉼표로 구분).

이러한 옵션에 대해 자세히 알아보려면 **man rhsm.conf**를 실행합니다.

Red Hat Subscription Manager 프록시 방식에는 다음과 같은 제한 사항이 있습니다.

- 이 방법은 Red Hat Subscription Manager에만 프록시 지원을 제공합니다.
- Red Hat Subscription Manager 프록시 구성 값은 시스템 전체 환경 변수에 대해 설정된 모든 값을 재정의합니다.

투명 프록시

네트워크에서 투명 프록시를 사용하여 애플리케이션 계층 트래픽을 관리하는 경우 프록시 관리가 자동으로 이루어지므로 프록시와 상호 작용하도록 언더클라우드 자체를 구성할 필요가 없습니다. 투명 프록시를 사용하면 Red Hat OpenStack Platform의 클라이언트 기반 프록시 구성에 따르는 제한 사항을 극복할 수 있습니다.

4.2. STACK 사용자 생성

director 설치 프로세스에는 root가 아닌 사용자가 명령을 실행해야 합니다. 다음 절차에 따라 **stack** 이라는 사용자를 생성하고 암호를 설정합니다.

절차

1. **root** 사용자로 언더클라우드에 로그인합니다.
2. **stack** 사용자를 생성합니다.

```
[root@director ~]# useradd stack
```

3. 사용자 암호를 설정합니다.

```
[root@director ~]# passwd stack
```

4. **sudo** 사용 시 암호를 요구하지 않도록 비활성화합니다.

```
[root@director ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a /etc/sudoers.d/stack
[root@director ~]# chmod 0440 /etc/sudoers.d/stack
```

5. 새로 만든 **stack** 사용자로 전환합니다.

```
[root@director ~]# su - stack
[stack@director ~]$
```

director 설치를 **stack** 사용자로 계속합니다.

4.3. 템플릿 및 이미지용 디렉터리 생성

director는 시스템 이미지와 Heat 템플릿을 사용하여 오버클라우드 환경을 생성합니다. 이러한 파일을 정리하여 보관하려면 이미지 및 템플릿에 대한 디렉터리를 생성하는 것이 좋습니다.

```
[stack@director ~]$ mkdir ~/images
[stack@director ~]$ mkdir ~/templates
```

4.4. 언더클라우드 호스트 이름 설정

언더클라우드에는 설치 및 구성 프로세스를 위해 정규화된 도메인 이름이 필요합니다. 사용하는 DNS 서버는 정규화된 도메인 이름을 확인할 수 있어야 합니다. 예를 들어 내부 또는 프라이빗 DNS 서버를 사용할 수 있습니다. 즉, 언더클라우드의 호스트 이름을 설정해야 할 수 있습니다.

절차

1. 언더클라우드의 기본 및 전체 호스트 이름을 확인합니다.

```
[stack@director ~]$ hostname
[stack@director ~]$ hostname -f
```

2. 이전 명령에서 올바른 정규화된 호스트 이름이 출력되지 않거나 오류가 나타나는 경우 **hostnamectl**을 사용하여 호스트 이름을 설정합니다.

```
[stack@director ~]$ sudo hostnamectl set-hostname manager.example.com
[stack@director ~]$ sudo hostnamectl set-hostname --transient manager.example.com
```

3. director에는 **/etc/hosts** 에 시스템의 호스트 이름 및 기본 이름도 입력해야 합니다. **/etc/hosts**의 IP 주소는 언더클라우드 공용 API에 사용할 주소와 일치해야 합니다. 예를 들어 시스템 이름이 **manager.example.com**이고 IP 주소로 **10.0.0.1**을 사용하는 경우 **/etc/hosts**에 다음과 같은 항목이 필요합니다.

```
10.0.0.1  manager.example.com manager
```

4.5. 언더클라우드 등록 및 업데이트

사전 요구 사항

director를 설치하기 전에 다음 작업을 완료합니다.

- Red Hat Subscription Manager에 언더클라우드 등록
- 관련 리포지토리를 구독하고 활성화합니다.
- Red Hat Enterprise Linux 패키지 업데이트 수행

절차

1. Content Delivery Network에 시스템을 등록합니다. 메시지가 표시되면 고객 포털 사용자 이름과 암호를 입력합니다.

```
[stack@director ~]$ sudo subscription-manager register
```

2. Red Hat OpenStack Platform director의 인타이틀먼트 풀 ID를 검색합니다. 예를 들면 다음과 같습니다.

```
[stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
                     Red Hat Virtualization
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             Valid-Pool-Number-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

3. **Pool ID** 값을 찾아서 Red Hat OpenStack Platform 13 인타이틀먼트를 연결합니다.

```
[stack@director ~]$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

4. 기본 리포지토리를 모두 비활성화한 다음 director 설치에 필요한 패키지가 포함된 필수 Red Hat Enterprise Linux 리포지토리를 활성화합니다.

```
[stack@director ~]$ sudo subscription-manager repos --disable=*
[stack@director ~]$ sudo subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-7-server-extras-rpms --enable=rhel-7-server-rh-common-rpms --enable=rhel-ha-for-rhel-7-server-rpms --enable=rhel-7-server-openstack-13-rpms --enable=rhel-7-server-rhceph-3-tools-rpms
```

-

중요

2.5절. “리포지토리 요구 사항” 에 나열된 리포지토리만 활성화합니다. 패키지와 소프트웨어 충돌이 발생할 수 있으므로 추가 리포지토리를 활성화하지 마십시오.

5. 시스템에서 업데이트를 실행하여 기본 시스템 패키지가 최신 상태인지 확인합니다.

```
[stack@director ~]$ sudo yum update -y
```

6. 시스템을 재부팅합니다.

```
[stack@director ~]$ sudo reboot
```

이제 director를 설치할 준비가 되었습니다.

4.6. DIRECTOR 패키지 설치

다음 절차에서는 Red Hat OpenStack Platform director와 관련된 패키지를 설치합니다.

절차

1. director 설치 및 설정에 필요한 명령행 툴을 설치합니다.

```
[stack@director ~]$ sudo yum install -y python-tripleoclient
```

4.7. CEPH-ANIBLE 설치

Red Hat OpenStack Platform에서 Ceph Storage를 사용하는 경우 **ceph-ansible** 패키지가 필요합니다.

Red Hat Ceph Storage를 사용하거나 배포에서 외부 Ceph Storage 클러스터를 사용하는 경우 **ceph-anible** 패키지를 설치하십시오. 기존 Ceph Storage 클러스터와의 통합에 대한 자세한 내용은 Integrating an Overcloud with an Existing Red Hat Ceph Cluster를 참조하십시오.

절차

1. 다음과 같이 Ceph 툴 리포지토리를 활성화합니다.

```
[stack@director ~]$ sudo subscription-manager repos --enable=rhel-7-server-rhceph-3-tools-rpms
```

2. **ceph-ansible** 패키지를 설치합니다.

```
[stack@director ~]$ sudo yum install -y ceph-ansible
```

4.8. DIRECTOR 설정

director 설치 프로세스에는 네트워크 구성을 결정하는 특정 설정이 필요합니다. 이 설정은 **stack** 사용자의 홈 디렉터리에 있는 템플릿에 **undercloud.conf** 로 저장됩니다. 다음 절차에서는 기본 템플릿을 기준으로 구성하는 방법을 설명합니다.

절차

1. Red Hat은 설치에 필요한 설정을 결정하는 데 도움이 되는 기본 템플릿을 제공합니다. 이 템플릿을 **stack** 사용자의 홈 디렉터리에 복사합니다.

```
[stack@director ~]$ cp \
  /usr/share/instack-undercloud/undercloud.conf.sample \
  ~/undercloud.conf
```

2. **undercloud.conf** 파일을 편집합니다. 이 파일에는 언더클라우드를 구성하는 설정이 포함되어 있습니다. 매개변수를 생략하거나 주석으로 처리하면 언더클라우드 설치에 기본값이 사용됩니다.

4.9. DIRECTOR 설정 매개변수

undercloud.conf 파일을 구성하기 위한 매개 변수 목록은 다음과 같습니다. 오류를 방지하려면 모든 매개변수를 관련 섹션에 보관합니다.

기본값

다음 매개변수는 **undercloud.conf** 파일의 **[DEFAULT]** 섹션에 정의됩니다.

undercloud_hostname

언더클라우드에 대해 정규화된 호스트 이름을 정의합니다. 설정되어 있는 경우 언더클라우드 설치 시 모든 시스템의 호스트 이름이 설정됩니다. 설정되어 있지 않은 경우 언더클라우드에서 현재 호스트 이름을 사용하지만 사용자가 모든 시스템의 호스트 이름을 적절하게 설정해야 합니다.

local_ip

director의 프로비저닝 NIC에 대해 정의된 IP 주소입니다. director에서 해당 DHCP 및 PXE 부팅 서비스에 사용하는 IP 주소이기도 합니다. 프로비저닝 네트워크에 다른 서브넷을 사용하지 않는 경우, 예를 들어 환경의 기존 IP 주소 또는 서브넷과 충돌하는 경우 이 값을 기본값 **192.168.24.1/24** 로 유지하십시오.

undercloud_public_host

SSL/TLS를 통한 director Public API 엔드포인트에 대해 정의된 IP 주소 또는 호스트 이름입니다. director 설정에 따라 **/32** 넷마스크를 사용하는 라우팅된 IP 주소로 IP 주소를 director 소프트웨어 브릿지에 연결합니다.

undercloud_admin_host

SSL/TLS를 통한 director Admin API 엔드포인트에 대해 정의된 IP 주소 또는 호스트 이름입니다. director 설정에 따라 **/32** 넷마스크를 사용하는 라우팅된 IP 주소로 IP 주소를 director 소프트웨어 브릿지에 연결합니다.

undercloud_nameservers

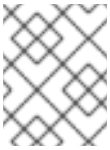
언더클라우드 호스트 이름 확인에 사용할 DNS 이름 서버 목록입니다.

undercloud_ntp_servers

언더클라우드의 날짜 및 시간을 동기화하는 데 사용되는 네트워크 시간 프로토콜 서버 목록입니다.

overcloud_domain_name

오버클라우드 배포 시 사용할 DNS 도메인 이름입니다.



참고

오버클라우드를 설정할 때 **CloudDomain** 매개변수를 일치하는 값으로 설정해야 합니다. 오버클라우드 구성 시 환경 파일에서 이 매개변수를 설정하십시오.

subnets

프로비저닝 및 인트로스펙션에 사용되는 라우팅된 네트워크 서브넷 목록입니다. 자세한 내용은 서브넷를 참조하십시오. 기본값은 **ctlplane-subnet** 서브넷만 포함합니다.

local_subnet

PXE 부팅 및 DHCP 인터페이스에 사용되는 로컬 서브넷입니다. **local_ip** 주소는 이 서브넷에 존재해야 합니다. 기본값은 **ctlplane-subnet**입니다.

masquerade_network

undercloud.conf 파일의 **[ctlplane-subnet]** 섹션에서 **masquerade: true** 를 설정하는 경우 **masquerade_network** 매개변수에 빈 값을 정의해야 합니다.

undercloud_service_certificate

OpenStack SSL/TLS 통신을 위한 인증서 위치 및 파일 이름입니다. 이 인증서를 신뢰할 수 있는 인증기관에서 가져오는 것이 가장 좋습니다. 그렇지 않으면 부록 A. *SSL/TLS 인증서 구성* 의 지침을 사용하여 자체 서명 인증서를 생성합니다. 이러한 지침에는 자체 서명된 인증서이든 권한이 있는지 여부에 관계없이 인증서에 대한 SELinux 컨텍스트를 설정하는 방법도 포함되어 있습니다. 이 옵션은 오버클라우드를 배포할 때 영향을 미칩니다. 자세한 내용은 6.9절. "언더클라우드 CA를 신뢰하도록 오버클라우드 노드 구성"를 참조하십시오.

generate_service_certificate

언더클라우드 설치 중에 **undercloud_service_certificate** 매개변수에 사용되는 SSL/TLS 인증서를 생성할지 여부를 정의합니다. 언더클라우드 설치에서 생성된 인증서 **/etc/pki/tls/certs/undercloud-[undercloud_public_vip].pem**을 저장합니다. **certificate_generation_ca** 매개변수에 정의된 CA가 이 인증서에 서명합니다. 이 옵션은 오버클라우드를 배포할 때 영향을 미칩니다. 자세한 내용은 6.9절. "언더클라우드 CA를 신뢰하도록 오버클라우드 노드 구성"를 참조하십시오.

certificate_generation_ca

요청된 인증서에 서명하는 CA의 **certmonger** 닉네임입니다. **generate_service_certificate** 매개변수를 설정한 경우에만 이 옵션을 사용합니다. **로컬** CA를 선택하는 경우 certmonger는 로컬 CA 인증서를 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** 으로 추출하고 신뢰 체인에 추가합니다.

service_principal

인증서를 사용하는 서비스에 대한 Kerberos 사용자입니다. FreeIPA에서와 같이 CA에 Kerberos 사용자가 필요한 경우에만 사용합니다.

local_interface

director의 프로비저닝 NIC용으로 선택한 인터페이스로, director에서 해당 DHCP 및 PXE 부팅 서비스에 사용하는 장치이기도 합니다. 이 값을 원하는 장치로 변경하십시오. 연결된 장치를 확인하려면 **ip addr** 명령을 사용합니다. 예를 들면 다음은 **ip addr** 명령을 실행한 결과입니다.

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic eth0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

이 예제에서 외부 NIC는 **eth0**을 사용하고, 프로비저닝 NIC는 현재 구성되지 않은 **eth1**을 사용합니다. 이 경우에는 **local_interface**를 **eth1**로 설정합니다. 설정 스크립트는 이 인터페이스를 **inspection_interface** 매개변수로 정의된 사용자 브릿지에 연결합니다.

local_mtu

local_interface에 사용할 MTU입니다. 언더클라우드의 경우 1500을 초과하지 마십시오.

hieradata_override

director에서 Puppet hieradata를 구성하여 **undercloud.conf** 매개변수 이외의 사용자 지정 설정을 서비스에 제공하는 **hieradata** 오버라이드 파일의 경로입니다. 설정한 경우 언더클라우드 설치 시 이 파일이 **/etc/puppet/hieradata** 디렉터리에 복사되고 계층에서 첫 번째 파일로 설정됩니다. 이 기능 사용에 대한 자세한 내용은 4.10절. "언더클라우드에서 hieradata 구성" 을 참조하십시오.

net_config_override

네트워크 구성 덮어쓰기 템플릿의 경로입니다. 설정되어 있는 경우 언더클라우드는 JSON 포멧 템플릿을 사용하여 네트워킹을 **os-net-config** 로 설정합니다. 이렇게 하면 **undercloud.conf** 에 설정된 네트워크 매개 변수가 무시됩니다. 본딩을 구성하거나 인터페이스에 옵션을 추가하려는 경우 이 매개변수를 사용합니다. **/usr/share/instack-undercloud/templates/net-config.json.template**예를 참조하십시오.

inspection_interface

director에서 노드 인트로스펙션에 사용하는 브릿지입니다. 이 브릿지는 director 구성으로 생성되는 사용자 지정 브릿지입니다. **LOCAL_INTERFACE**가 이 브릿지에 연결됩니다. 이 브릿지를 기본값 **br-ctlplane**으로 두십시오.

inspection_extras

검사 프로세스 중에 추가 하드웨어 컬렉션의 활성화 여부를 정의합니다. 인트로스펙션 이미지에 **python-hardware** 또는 **python-hardware-detect** 패키지가 필요합니다.

inspection_runbench

노드 인트로스펙션 중 벤치마크 집합을 실행합니다. 활성화하려면 **true** 로 설정합니다. 등록된 노드의 하드웨어를 검사할 때 벤치마크 분석을 수행하려는 경우 이 옵션이 필요합니다. 자세한 내용은 6.2절. "노드의 하드웨어 검사"를 참조하십시오.

inspection_enable_uefi

UEFI 전용 펌웨어가 있는 노드의 인트로스펙션을 지원할지 여부를 정의합니다. 자세한 내용은 부록 D. *대체 부팅 모드*의 내용을 참조하십시오.

enable_node_discovery

인트로스펙션 램디스크를 PXE 부팅하는 알려지지 않은 노드를 자동으로 등록합니다. 새로운 노드는 **fake_pxe** 드라이버를 기본값으로 사용하지만 덮어쓸 **discovery_default_driver**를 설정할 수 있습니다. 또한 introspection 규칙을 사용하여 새로 등록된 노드의 드라이버 정보를 지정할 수 있습니다.

discovery_default_driver

자동으로 등록된 노드의 기본 드라이버를 설정합니다. **enable_node_discovery** 를 활성화해야 하며 드라이버를 **enabled_drivers** 목록에 포함해야 합니다. 지원되는 드라이버 목록은 부록 B. *전원 관리 드라이버* 을 참조하십시오.

undercloud_debug

언더클라우드 서비스의 로그 수준을 **DEBUG**로 설정합니다. 활성화하려면 이 값을 **true**로 설정합니다.

undercloud_update_packages

언더클라우드 설치 중 패키지 업데이트 여부를 정의합니다.

enable_tempest

유효성 검사 툴 설치 여부를 정의합니다. 기본값은 **false** 로 설정되지만 **true** 를 사용하여 활성화할 수 있습니다.

enable_telemetry

언더클라우드에 OpenStack Telemetry 서비스(ceilometer, aodh, panko, gnocchi) 설치 여부를 정의합니다. Red Hat OpenStack Platform에서 Telemetry의 지표 백엔드는 gnocchi에서 제공합니다. **enable_telemetry** 매개변수를 **true** 로 설정하면 Telemetry 서비스가 자동으로 설치 및 설정됩니다. 기본값은 **false**로, 언더클라우드에서 Telemetry를 비활성화합니다. 이 매개변수는 Red Hat CloudForms와 같은 메트릭 데이터를 사용하는 기타 제품을 사용하는 경우에 필요합니다.

enable_ui

director의 웹 UI 설치 여부를 정의합니다. 이를 통해 그래픽 웹 인터페이스를 통해 오버클라우드 플래닝 및 배포를 수행할 수 있습니다. 자세한 내용은 *7장. 웹 UI로 기본 오버클라우드 구성*의 내용을 참조하십시오. UI는 **undercloud_service_certificate** 또는 **generate_service_certificate** 를 사용하여 활성화된 SSL/TLS에서만 사용할 수 있습니다.

enable_validations

검증을 실행하는 요구 사항 설치 여부를 정의합니다.

enable_novajoin

언더클라우드에서 **novajoin** 메타데이터 서비스 설치 여부를 정의합니다.

ipa_otp

언더클라우드 노드를 IPA 서버에 등록할 때 사용할 일회성 암호를 정의합니다. 이 암호는 **enable_novajoin**이 활성화된 경우 필요합니다.

ipxe_enabled

iPXE 또는 표준 PXE 사용 여부를 정의합니다. 기본값은 **true**이며, iPXE를 활성화합니다. 표준 PXE로 설정하려면 **false**로 설정합니다. 자세한 내용은 *부록 D. 대체 부팅 모드*의 내용을 참조하십시오.

scheduler_max_attempts

스케줄러가 인스턴스 배포를 시도하는 최대 횟수입니다. 스케줄링할 때 잠재적인 경합 조건을 해결하기 위해 즉시 배포해야 하는 베어 메탈 노드 수보다 크거나 같게 유지합니다.

clean_nodes

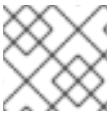
배포 중에 그리고 인트로스펙션 후에 하드 드라이브를 초기화할 것인지 여부를 정의합니다.

enabled_hardware_types

언더클라우드에 사용할 하드웨어 유형 목록입니다. 지원되는 드라이버 목록은 *부록 B. 전원 관리 드라이버* 을 참조하십시오.

additional_architectures

오버클라우드에서 지원할 (커널) 아키텍처 목록입니다. 현재는 **ppc64le**로 제한됩니다.



참고

ppc64le에 대한 지원을 사용하면 **ipxe_enabled**를 **False**로 설정해야 합니다.

암호

다음 매개변수는 **undercloud.conf** 파일의 **[auth]** 섹션에 정의됩니다.

undercloud_db_password, undercloud_admin_token, undercloud_admin_password, undercloud_glance_password; 등

나머지 매개변수는 director의 모든 서비스에 대한 액세스 세부 정보입니다. 값에는 변경이 필요하지 않습니다. **undercloud.conf** 에 비어 있는 경우 director의 설정 스크립트에서 이러한 값을 자동으로 생성합니다. 구성 스크립트가 완료된 후 모든 값을 검색할 수 있습니다. 이러한 암호의 영숫자 값만 특수 문자로 사용하면 구문 오류가 발생할 수 있습니다.

중요

이러한 매개 변수의 구성 파일 예제에서는 자리 표시자 값으로 **<None>** 을 사용합니다. 이러한 값을 **<None>** 으로 설정하면 배포 오류가 발생합니다.

서브넷

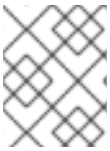
각 프로비저닝 서브넷은 **undercloud.conf** 파일에서 이름이 지정된 섹션입니다. 예를 들어 **ctlplane-subnet** 이라는 서브넷을 생성하려면 다음을 실행합니다.

```
[ctlplane-subnet]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.5
dhcp_end = 192.168.24.24
inspection_iprange = 192.168.24.100,192.168.24.120
gateway = 192.168.24.1
masquerade = true
```

환경에 따라 필요한 만큼의 프로비저닝 네트워크를 지정할 수 있습니다.

gateway

오버클라우드 인스턴스의 게이트웨이입니다. 트래픽을 외부 네트워크로 전달하는 언더클라우드 호스트입니다. director에 다른 IP 주소를 사용하거나 외부 게이트웨이를 직접 사용하려는 경우를 제외하고 기본값 **192.168.24.1** 로 두십시오.



참고

director의 설정 스크립트는 적절한 **sysctl** 커널 매개 변수를 사용하여 IP 포워딩을 자동으로 활성화합니다.

cidr

director에서 오버클라우드 인스턴스를 관리하는 데 사용하는 네트워크입니다. 이 네트워크는 언더클라우드의 **neutron** 서비스에서 관리하는 프로비저닝 네트워크입니다. 프로비저닝 네트워크에 다른 서브넷을 사용하지 않는 경우 기본값 **192.168.24.0/24** 로 두십시오.

masquerade

외부 액세스를 위해 **cidr**에 정의된 네트워크를 마스커레이드할지 여부를 정의합니다. 그러면 프로비저닝 네트워크에 일정 수준의 NAT(네트워크 주소 변환)가 제공되어 director를 통해 외부 액세스가 가능합니다. **masquerade** 매개변수를 **true** 로 설정하는 경우 **undercloud.conf** 파일의 **[DEFAULT]** 섹션에 **masquerade_network** 매개변수에 빈 masquerade 네트워크 값을 정의해야 합니다.

dhcp_start; dhcp_end

오버클라우드 노드의 DHCP 할당 범위 시작과 끝 값입니다. 이 범위에 노드를 할당할 충분한 IP 주소가 포함되어 있는지 확인하십시오.

inspection_iprange

director의 인트로스펙션 서비스에서 PXE 부팅 및 프로비저닝 프로세스 중에 사용하는 IP 주소 범위입니다. 쉼표로 구분된 값을 사용하여 이 범위의 시작 및 끝을 정의합니다. 예: **192.168.24.100,192.168.24.120**. 이 범위에 노드에 충분한 IP 주소가 포함되어 있고 **dhcp_start** 및 **dhcp_end** 의 범위와 충돌하지 않는지 확인합니다.

이러한 매개변수의 값을 설정에 맞게 수정합니다. 완료되면 파일을 저장합니다.

4.10. 언더클라우드에서 HIERADATA 구성

director에서 Puppet hieradata를 설정하여 사용 가능한 **undercloud.conf** 매개변수 이외의 사용자 지정 설정을 서비스에 제공할 수 있습니다. 이 기능을 사용하려면 다음 절차를 수행합니다.

절차

1. **/home/stack/hieradata.yaml**과 같은 hieradata 오버라이드 파일을 생성합니다.

2. 파일에 사용자 지정 hieradata를 추가합니다. 예를 들어 Compute(nova) 서비스 매개변수 **force_raw_images**를 기본값 "True"에서 "False"로 수정하려면 다음을 추가합니다.

```
nova::compute::force_raw_images: False
```

설정할 매개변수에 대한 Puppet 구현이 없는 경우 다음 방법을 사용하여 매개변수를 설정합니다.

```
nova::config::nova_config:
  DEFAULT/<parameter_name>:
    value: <parameter_value>
```

예를 들면 다음과 같습니다.

```
nova::config::nova_config:
  DEFAULT/network_allocate_retries:
    value: 20
  ironic/serial_console_state_timeout:
    value: 15
```

3. **undercloud.conf**에서 **hieradata_override** 매개변수를 hieradata 파일의 경로로 설정합니다.

```
hieradata_override = /home/stack/hieradata.yaml
```

4.11. DIRECTOR 설치

다음 절차에서는 director를 설치하고 몇 가지 기본적인 설치 후 작업을 수행합니다.

절차

1. 다음 명령을 실행하여 언더클라우드에 director를 설치합니다.

```
[stack@director ~]$ openstack undercloud install
```

이 명령을 수행하면 director의 설정 스크립트가 실행됩니다. director가 추가 패키지를 설치하고, **undercloud.conf** 의 설정에 맞게 해당 서비스를 구성합니다. 이 스크립트를 완료하는 데 몇 분이 걸립니다.

스크립트가 완료되면 다음 두 파일이 생성됩니다.

- **undercloud-passwords.conf** - director의 서비스에 대한 모든 암호 목록입니다.
- **stackrc** - director의 명령줄 툴에 액세스할 수 있도록 지원하는 초기화 변수 세트입니다.

2. 이 스크립트는 모든 OpenStack Platform 서비스를 자동으로 시작합니다. 다음 명령을 사용하여 활성화된 서비스를 확인합니다.

```
[stack@director ~]$ sudo systemctl list-units openstack-*
```

3. 이 스크립트는 **stack** 사용자를 **docker** 그룹에 추가하여 **stack** 사용자에게 컨테이너 관리 명령에 대한 액세스 권한을 부여합니다. 다음 명령을 사용하여 **stack** 사용자의 권한을 새로 고칩니다.

```
[stack@director ~]$ exec su -l stack
```

명령을 실행하면 다시 로그인하라는 메시지가 표시됩니다. stack 사용자의 암호를 입력합니다.

4. **stack** 사용자를 초기화하여 명령줄 툴을 사용하려면 다음 명령을 실행합니다.

```
[stack@director ~]$ source ~/stackrc
```

프롬프트 메세지에 OpenStack 명령이 언더클라우드를 인증 및 실행될 수 있음을 나타냅니다.

```
(undercloud) [stack@director ~]$
```

director 설치가 완료되었습니다. 이제 director의 명령행 툴을 사용할 수 있습니다.

4.12. 오버클라우드 노드의 이미지 가져오기

director에는 오버클라우드 노드를 프로비저닝하기 위한 여러 개의 디스크 이미지가 필요합니다. 여기에는 다음이 포함됩니다.

- 인트로스펙션 커널 및 램디스크 - PXE 부팅을 통해 베어 메탈 시스템의 인트로스펙션에 사용됩니다.
- 배포 커널 및 램디스크 - 시스템 프로비저닝 및 배포에 사용됩니다.
- 오버클라우드 커널, 램디스크 및 전체 이미지 - 노드의 하드 디스크에 기록된 기본 오버클라우드 시스템입니다.

다음 절차는 오버클라우드 노드의 이미지를 확보하고 설치하는 방법을 설명합니다.

4.12.1. 단일 CPU 아키텍처 오버클라우드

기본 CPU 아키텍처 x86-64를 사용하는 오버클라우드를 배포하려면 다음 이미지 및 절차가 필요합니다.

절차

1. **stackrc** 파일을 소싱하여 director의 명령행 툴을 활성화합니다.

```
[stack@director ~]$ source ~/stackrc
```

2. **rhosp-director-images** 및 **rhosp-director-images-ipa** 패키지를 설치합니다.

```
(undercloud) [stack@director ~]$ sudo yum install rhosp-director-images rhosp-director-images-ipa
```

3. **stack** 사용자 홈(**/home/stack/images**)의 **images** 디렉터리에 압축된 이미지 파일을 풉니다.

```
(undercloud) [stack@director ~]$ mkdir ~/images
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-13.0.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-13.0.tar; do tar -xvf $i; done
```

4. 이러한 이미지를 director로 가져옵니다.

```
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
/home/stack/images/
```

이렇게 하면 다음 이미지가 director에 업로드됩니다.

- **bm-deploy-kernel**
- **bm-deploy-ramdisk**
- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

이 스크립트는 director의 PXE 서버에 인트로스펙션 이미지도 설치합니다.

5. 이미지가 성공적으로 업로드되었는지 확인합니다.

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 765a46af-4417-4592-91e5-a300ead3faf6 | bm-deploy-ramdisk      |
| 09b40e3d-0382-4925-a356-3a4b4f36b514 | bm-deploy-kernel       |
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

이 목록에는 인트로스펙션 PXE 이미지가 표시되지 않습니다. director는 이 파일을 **/httpboot** 에 복사합니다.

```
(undercloud) [stack@director images]$ ls -l /httpboot
total 341460
-rwxr-xr-x. 1 root             root               5153184 Mar 31 06:58 agent.kernel
-rw-r--r--. 1 root             root             344491465 Mar 31 06:59 agent.ramdisk
-rw-r--r--. 1 ironic-inspector  ironic-inspector        337 Mar 31 06:23 inspector.ipxe
```

4.12.2. 여러 CPU 아키텍처 오버클라우드

다음은 추가 CPU 아키텍처를 지원하는 오버클라우드 배포에 필요한 이미지 및 절차입니다. 현재는 ppc64le, Power Architecture로 제한됩니다.

절차

1. **stackrc** 파일을 소싱하여 director의 명령행 툴을 활성화합니다.

```
[stack@director ~]$ source ~/stackrc
```

2. **rhosp-director-images-all** 패키지를 설치합니다.

```
(undercloud) [stack@director ~]$ sudo yum install rhosp-director-images-all
```

3. **stack** 사용자 홈(**/home/stack/images**)의 **images** 디렉터리에 있는 아키텍처별 디렉터리에 아카이브를 풉니다.

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do mkdir $arch ; done
(undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do for i in
/usr/share/rhosp-director-images/overcloud-full-latest-13.0-${arch}.tar /usr/share/rhosp-
director-images/ironic-python-agent-latest-13.0-${arch}.tar ; do tar -C $arch -xf $i ; done ;
done
```

4. 이러한 이미지를 director로 가져옵니다.

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/ppc64le --architecture ppc64le --whole-disk --http-boot /tftpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64/ --http-boot /tftpboot
```

이렇게 하면 다음 이미지가 director에 업로드됩니다.

- **bm-deploy-kernel**
- **bm-deploy-ramdisk**
- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**
- **ppc64le-bm-deploy-kernel**
- **ppc64le-bm-deploy-ramdisk**
- **ppc64le-overcloud-full**
  이 스크립트는 director PXE 서버에 인트로스펙션 이미지도 설치합니다.

5. 이미지가 성공적으로 업로드되었는지 확인합니다.

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+--------+
| ID                                   | Name                    | Status |
+--------------------------------------+-------------------------+--------+
| 6d1005ba-ec82-473b-8e33-88aadb5b6792 | bm-deploy-kernel        | active |
| fb723b33-9f11-45f5-b25b-c008bf509290 | bm-deploy-ramdisk       | active |
| 6a6096ba-8f79-4343-b77c-4349f7b94960 | overcloud-full          | active |
| de2a1bde-9351-40d2-bbd7-7ce9d6eb50d8 | overcloud-full-initrd   | active |
| 67073533-dd2a-4a95-8e8b-0f108f031092 | overcloud-full-vmlinuz  | active |
| 69a9ffe5-06dc-4d81-a122-e5d56ed46c98 | ppc64le-bm-deploy-kernel | active |
| 464dd809-f130-4055-9a39-cf6b63c1944e | ppc64le-bm-deploy-ramdisk | active |
| f0fedcd0-3f28-4b44-9c88-619419007a03 | ppc64le-overcloud-full  | active |
+--------------------------------------+-------------------------+--------+
```

이 목록에는 인트로스펙션 PXE 이미지가 표시되지 않습니다. director는 이 파일을 **/tftpboot**에 복사합니다.

```
(undercloud) [stack@director images]$ ls -l /tftpboot /tftpboot/ppc64le/
/tftpboot:
total 422624
-rwxr-xr-x. 1 root   root     6385968 Aug  8 19:35 agent.kernel
-rw-r--r--. 1 root   root   425530268 Aug  8 19:35 agent.ramdisk
-rwxr--r--. 1 ironic ironic     20832 Aug  8 02:08 chain.c32
-rwxr--r--. 1 ironic ironic    715584 Aug  8 02:06 ipxe.efi
-rw-r--r--. 1 root   root          22 Aug  8 02:06 map-file
drwxr-xr-x. 2 ironic ironic        62 Aug  8 19:34 ppc64le
-rwxr--r--. 1 ironic ironic     26826 Aug  8 02:08 pxelinux.0
drwxr-xr-x. 2 ironic ironic        21 Aug  8 02:06 pxelinux.cfg
-rwxr--r--. 1 ironic ironic     69631 Aug  8 02:06 undionly.kpxe

/tftpboot/ppc64le/:
total 457204
-rwxr-xr-x. 1 root             root             19858896 Aug  8 19:34 agent.kernel
-rw-r--r--. 1 root             root            448311235 Aug  8 19:34 agent.ramdisk
-rw-r--r--. 1 ironic-inspector ironic-inspector       336 Aug  8 02:06 default
```

4.12.3. 최소 오버클라우드 이미지

overcloud-minimal 이미지를 사용하여 다른 Red Hat OpenStack Platform 서비스를 실행하지 않으려는 베어 OS를 프로비저닝하거나 서브스크립션 인타이틀먼트 중 하나를 사용할 수 있습니다.

절차

1. **stackrc** 파일을 소싱하여 director 명령행 툴을 활성화합니다.

```
[stack@director ~]$ source ~/stackrc
```

2. **overcloud-minimal** 패키지를 설치합니다.

```
(undercloud) [stack@director ~]$ sudo yum install rhosp-director-images-minimal
```

3. **stack** 사용자의 홈 디렉터리(**/home/stack/images**)에 있는 **images** 디렉터리에 압축된 이미지 파일을 풉니다.

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ tar xf /usr/share/rhosp-director-images/overcloud-minimal-latest-13.0.tar
```

4. 이미지를 director로 가져옵니다.

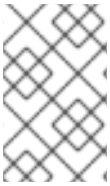
```
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path /home/stack/images/ --os-image-name overcloud-minimal.qcow2
```

이 스크립트는 다음 이미지를 director에 업로드합니다.

- **overcloud-minimal**
- **overcloud-minimal-initrd**
- **overcloud-minimal-vmlinuz**

5. 이미지가 성공적으로 업로드되었는지 확인합니다.

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
| 32cf6771-b5df-4498-8f02-c3bd8bb93fdd | overcloud-minimal      |
| 600035af-dbbb-4985-8b24-a4e9da149ae5 | overcloud-minimal-initrd  |
| d45b0071-8006-472b-bbcc-458899e0d801 | overcloud-minimal-vmlinuz |
+--------------------------------------+------------------------+
```



참고

기본 **overcloud-full.qcow2** 이미지는 플랫 파티션 이미지입니다. 하지만 전체 디스크 이미지를 가져와서 사용할 수도 있습니다. 자세한 내용은 부록 C. *전체 디스크 이미지*를 참조하십시오.

4.13. 컨트롤 플레인의 네임서버 설정

오버클라우드에서 **cdn.redhat.com**과 같은 외부 호스트 이름을 확인하도록 하려면 오버클라우드 노드에 네임서버를 설정하는 것이 좋습니다. 네트워크를 분리하지 않은 표준 오버클라우드의 경우 언더클라우드의 컨트롤 플레인 서브넷을 사용하여 네임서버가 정의됩니다. 다음 절차에 따라 환경에 대한 네임서버를 정의하십시오.

절차

1. **stackrc** 파일을 소싱하여 director의 명령행 툴을 활성화합니다.

```
[stack@director ~]$ source ~/stackrc
```

2. **ctlplane-subnet** 서브넷의 네임서버를 설정합니다.

```
(undercloud) [stack@director images]$ openstack subnet set --dns-nameserver
[nameserver1-ip] --dns-nameserver [nameserver2-ip] ctlplane-subnet
```

각 네임서버에 대해 **--dns-nameserver** 옵션을 사용합니다.

3. 서브넷을 표시하여 네임 서버를 확인합니다.

```
(undercloud) [stack@director images]$ openstack subnet show ctlplane-subnet
+-------------------+-----------------------------------------+
| Field             | Value                                   |
+-------------------+-----------------------------------------+
| ...               |                                         |
| dns_nameservers   | 8.8.8.8                                 |
| ...               |                                         |
+-------------------+-----------------------------------------+
```

중요

서비스 트래픽을 별도의 네트워크에 분리하려면 오버클라우드 노드에서 네트워크 환경 파일에 **DnsServers** 매개변수를 사용합니다.

4.14. 다음 단계

이렇게 하면 director 설정 및 설치가 완료됩니다. 다음 장에서는 노드 등록, 검사 및 여러 노드 역할 태그 지정을 포함하여 오버클라우드의 기본적인 구성에 대해 설명합니다.

5장. 컨테이너 이미지 소스 구성

모든 오버클라우드 서비스는 컨테이너화되어 있으므로 오버클라우드에서 필요한 컨테이너 이미지를 사용하여 레지스트리에 액세스해야 합니다. 이 장에서는 Red Hat OpenStack Platform에 컨테이너 이미지를 사용하도록 레지스트리 및 오버클라우드 설정을 준비하는 방법에 대해 설명합니다.

- 이 가이드에서는 레지스트리를 사용하도록 오버클라우드를 구성하는 몇 가지 사용 사례를 제공합니다. 이러한 방법에 대한 설명은 5.1절. "레지스트리 방법" 을 참조하십시오.
- 이미지 준비 명령을 사용하는 방법을 숙지하는 것이 좋습니다. 자세한 내용은 5.2절. "컨테이너 이미지 준비 명령 사용"를 참조하십시오.
- 컨테이너 이미지 소스 준비에 가장 일반적인 방법을 사용하려면 5.5절. "언더클라우드를 로컬 레지스트리로 사용" 를 참조하십시오.

5.1. 레지스트리 방법

Red Hat OpenStack Platform은 다음과 같은 레지스트리 유형을 지원합니다.

원격 레지스트리

오버클라우드는 **registry.redhat.io** 에서 직접 컨테이너 이미지를 가져옵니다. 이 방법은 초기 구성을 생성하는 가장 쉬운 방법입니다. 그러나 각 오버클라우드 노드는 Red Hat Container Catalog에서 직접 각 이미지를 가져오므로 네트워크 혼잡이 발생하고 배포가 느려질 수 있습니다. 또한 모든 오버클라우드 노드에는 Red Hat Container Catalog에 인터넷 액세스가 필요합니다.

로컬 레지스트리

언더클라우드는 **docker-distribution** 서비스를 사용하여 레지스트리 역할을 합니다. 이렇게 하면 director에서 **registry.redhat.io** 의 이미지를 동기화하고 **docker-distribution** 레지스트리로 푸시할 수 있습니다. 오버클라우드를 생성할 때 오버클라우드는 언더클라우드의 **docker-distribution** 레지스트리에서 컨테이너 이미지를 가져옵니다. 이 방법을 사용하면 내부적으로 레지스트리를 저장할 수 있으므로 배포 속도를 높이고 네트워크 혼잡을 줄일 수 있습니다. 그러나 언더클라우드는 기본 레지스트리 역할을 하며 컨테이너 이미지에 대해 제한된 라이프 사이클 관리 기능을 제공합니다.

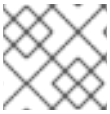
참고

docker-distribution 서비스는 docker . **docker** 와 별도로 작동합니다.**docker** 는 이미지를 **docker-distribution** 레지스트리로 가져오고 푸시하는 데 사용되며 오버클라우드 이미지를 제공하지 않습니다. 오버클라우드는 **docker-distribution** 레지스트리에서 이미지를 가져옵니다.

Satellite Server

컨테이너 이미지의 전체 애플리케이션 라이프사이클을 관리하고 Red Hat Satellite 6 서버를 통해 게시합니다. 오버클라우드는 Satellite 서버에서 이미지를 가져옵니다. 이 방법은 Red Hat OpenStack Platform 컨테이너를 저장, 관리 및 배포할 수 있는 엔터프라이즈급 솔루션을 제공합니다.

목록에서 방법을 선택하고 레지스트리 세부 정보를 계속 구성합니다.



참고

다중 아키텍처 클라우드의 빌드 시 로컬 레지스트리 옵션은 지원되지 않습니다.

5.2. 컨테이너 이미지 준비 명령 사용

이 섹션에서는 명령의 다양한 옵션에 대한 개념 정보를 포함하여 **openstack overcloud container image prepare** 명령을 사용하는 방법에 대한 개요를 제공합니다.

오버클라우드용 컨테이너 이미지 환경 파일 생성

openstack overcloud container image prepare 명령을 사용하는 주요 용도 중 하나는 오버클라우드에서 사용하는 이미지 목록이 포함된 환경 파일을 생성하는 것입니다. **openstack overcloud deploy** 와 같은 오버클라우드 배포 명령을 사용하여 이 파일을 추가합니다. **openstack overcloud container image prepare** 명령에서는 이 기능에 다음 옵션을 사용합니다.

--output-env-file

결과 환경 파일 이름을 정의합니다.

다음 스니펫은 이 파일 콘텐츠의 예입니다.

```
parameter_defaults:
  DockerAodhApiImage: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
  DockerAodhConfigImage: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
...
```

환경 파일에는 언더클라우드 레지스트리의 IP 주소 및 포트로 설정된 **DockerInsecureRegistryAddress** 매개변수도 포함되어 있습니다. 이 매개변수는 SSL/TLS 인증없이 언더클라우드 레지스트리에서 이미지에 액세스하도록 오버클라우드 노드를 구성합니다.

가져오기를 위한 컨테이너 이미지 목록 생성

OpenStack Platform 컨테이너 이미지를 다른 레지스트리 소스로 가져오려는 경우 이미지 목록을 생성할 수 있습니다. 목록의 구문은 주로 언더클라우드의 컨테이너 레지스트리로 컨테이너 이미지를 가져오는 데 사용되지만, Red Hat Satellite 6과 같은 다른 가져오기 방법에 맞게 이 목록의 형식을 수정할 수 있습니다.

openstack overcloud container image prepare 명령에서는 이 기능에 다음 옵션을 사용합니다.

--output-images-file

가져오기 목록의 결과 파일 이름을 정의합니다.

다음은 이 파일 콘텐츠의 예입니다.

```
container_images:
- imagename: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
- imagename: registry.redhat.io/rhosp13/openstack-aodh-evaluator:13.0-34
...
```

컨테이너 이미지의 네임스페이스 설정

--output-env-file 및 **--output-images-file** 옵션 모두 결과 이미지 위치를 생성하는 네임스페이스가 필요합니다. **openstack overcloud container image prepare** 명령에서는 다음 옵션을 사용하여 가져올 컨테이너 이미지의 소스 위치를 설정합니다.

--namespace

컨테이너 이미지의 네임스페이스를 정의합니다. 일반적으로 디렉터리가 있는 호스트 이름 또는 IP 주소입니다.

--prefix

이미지 이름 앞에 추가할 접두사를 정의합니다.

결과적으로 director는 다음 형식을 사용하여 이미지 이름을 생성합니다.

- **[NAMESPACE]/[PREFIX][IMAGE NAME]**

컨테이너 이미지 태그 설정

--tag 및 **--tag-from-label** 옵션을 함께 사용하여 각 컨테이너 이미지의 태그를 설정합니다.

--tag

소스의 모든 이미지에 대한 특정 태그를 설정합니다. 이 옵션만 사용하면 director가 이 태그를 사용하여 모든 컨테이너 이미지를 가져옵니다. 그러나 **--tag-from-label** 과 함께 이 옵션을 사용하는 경우 director는 **--tag** 를 소스 이미지로 사용하여 레이블을 기반으로 특정 버전 태그를 식별합니다. 기본적으로 **--tag** 옵션은 **latest** 로 설정됩니다.

--tag-from-label

지정된 컨테이너 이미지 레이블의 값을 사용하여 모든 이미지에 대해 버전이 지정된 태그를 검색하고 가져옵니다. director는 **--tag** 용으로 설정한 값으로 태그된 각 컨테이너 이미지를 검사한 다음 컨테이너 이미지 레이블을 사용하여 director가 레지스트리에서 가져오는 새 태그를 구성합니다. 예를 들어 **--tag-from-label {version}-{release}** 를 설정하면 director는 **version** 및 **release** 레이블을 사용하여 새 태그를 구성합니다. 한 컨테이너의 경우 **version** 을 **13.0** 으로 설정하고 **릴리스** 를 **34** 으로 설정할 수 있으므로 태그 **13.0-34** 이 됩니다.

중요

Red Hat Container Registry는 특정 버전 형식을 사용하여 모든 Red Hat OpenStack Platform 컨테이너 이미지에 태그를 지정합니다. 이 버전 형식은 **{version}-{release}** 이며, 각 컨테이너 이미지는 컨테이너 메타데이터에 레이블로 저장됩니다. 이 버전 형식은 하나의 **{release}** 에서 다음 버전으로의 업데이트를 원활하게 수행할 수 있습니다. 따라서 **openstack overcloud container image prepare** 명령을 실행할 때 항상 **--tag-from-label {version}-{release}** 를 사용해야 합니다. 컨테이너 이미지를 가져오는 데 자체적으로 **--tag** 를 사용하지 마십시오. 예를 들어 **--tag latest** 를 사용하면 director에서 컨테이너 이미지를 업데이트하기 위해 태그를 변경해야 하므로 업데이트를 수행할 때 문제가 발생합니다.

5.3. 추가 서비스를 위한 컨테이너 이미지

director는 핵심 OpenStack Platform 서비스에 대한 컨테이너 이미지만 준비합니다. 일부 추가 기능은 추가 컨테이너 이미지가 필요한 서비스를 사용합니다. 환경 파일을 사용하여 이러한 서비스를 활성화합니다. **openstack overcloud container image prepare** 명령은 다음 옵션을 사용하여 환경 파일 및 해당 컨테이너 이미지를 포함합니다.

-e

추가 컨테이너 이미지를 활성화하려면 환경 파일을 포함합니다.

다음 표에서는 컨테이너 이미지 및 **/usr/share/openstack-tripleo-heat-templates** 디렉터리 내의 해당 환경 파일 위치를 사용하는 추가 서비스의 샘플 목록을 제공합니다.

Service	환경 파일
Ceph Storage	**environments/ceph-ansible/ceph-ansible.yaml**
collectd	**environments/services-docker/collectd.yaml**
부서명	**environments/services-docker/congress.yaml**
fluentd	**environments/services-docker/fluentd.yaml**

Service	환경 파일
OpenStack Bare Metal(ironic)	**environments/services-docker/ironic.yaml**
OpenStack Data Processing(sahara)	**environments/services-docker/sahara.yaml**
OpenStack EC2-API	**environments/services-docker/ec2-api.yaml**
OpenStack Key Manager(barbican)	**environments/services-docker/barbican.yaml**
OpenStack Load Balancing-as-a-Service(octavia)	**environments/services-docker/octavia.yaml**
OpenStack Shared File System Storage(manila)	**environments/manila-{backend-name}-config.yaml** 알림: 자세한 내용은 OpenStack Shared File System(manila)을 참조하십시오.
OVN(Open Virtual Network)	**environments/services-docker/neutron-ovn-dvr-ha.yaml**
ensu	**environments/services-docker/sensu-client.yaml**

다음 몇 섹션에서는 추가 서비스를 포함하는 예를 제공합니다.

Ceph Storage

오버클라우드와 함께 Red Hat Ceph Storage 클러스터를 배포하는 경우 **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** 환경 파일을 포함해야 합니다. 이 파일을 사용하면 오버클라우드에서 구성 가능한 컨테이너화된 서비스를 사용할 수 있으며, 이미지를 준비하기 위해 director에서 이러한 서비스가 활성화되어 있는지 확인해야 합니다.

이 환경 파일 외에도 OpenStack Platform 서비스와 다른 Ceph Storage 컨테이너 위치를 정의해야 합니다. **--set** 옵션을 사용하여 Ceph Storage와 관련된 다음 매개변수를 설정합니다.

--set ceph_namespace

Ceph Storage 컨테이너 이미지의 네임스페이스를 정의합니다. 이 기능은 **--namespace** 옵션과 유사합니다.

--set ceph_image

Ceph Storage 컨테이너 이미지의 이름을 정의합니다. 일반적으로 **rhceph-3-rhel7** 입니다.

--set ceph_tag

Ceph Storage 컨테이너 이미지에 사용할 태그를 정의합니다. 이 기능은 **--tag** 옵션과 유사합니다. **--tag-from-label** 이 지정되면 이 태그부터 versioned 태그가 검색됩니다.

다음 스니펫은 컨테이너 이미지 파일에 Ceph Storage를 포함하는 예제입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml \
```

```
  --set ceph_namespace=registry.redhat.io/rhceph \
  --set ceph_image=rhceph-3-rhel7 \
  --tag-from-label {version}-{release} \
  ...
```

OpenStack Bare Metal(ironic)

오버클라우드에 OpenStack Bare Metal(ironic)을 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml** 환경 파일을 포함해야 합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml \
  ...
```

OpenStack Data Processing(sahara)

오버클라우드에 OpenStack Data Processing(sahara)을 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/sahara.yaml** 환경 파일을 포함해야 합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/sahara.yaml \
  ...
```

OpenStack Neutron SR-IOV

오버클라우드에 OpenStack Neutron SR-IOV를 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-sriov.yaml** 환경 파일을 포함합니다. 기본 컨트롤러 및 Compute 역할은 SR-IOV 서비스를 지원하지 않으므로 SR-IOV 서비스가 포함된 사용자 정의 역할 파일을 추가하려면 **-r** 옵션을 사용해야 합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -r ~/custom_roles_data.yaml
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-sriov.yaml \
  ...
```

OpenStack Load Balancing-as-a-Service(octavia)

오버클라우드에 OpenStack Load Balancing-as-a-Service를 배포하는 경우 director가 이미지를 준비할 수 있도록 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml** 환경 파일을 포함합니다. 다음 코드 조각은 이 환경 파일을 포함하는 방법에 대한 예입니다.

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml
\
  ...
```

OpenStack Shared File System(manila)

manila-{backend-name}-config.yaml 형식을 사용하여 지원되는 백엔드를 선택하여 해당 백엔드와 공유 파일 시스템을 배포할 수 있습니다. 공유 파일 시스템 서비스 컨테이너는 다음 환경 파일을 포함하여 준비할 수 있습니다.

```
environments/manila-isilon-config.yaml
environments/manila-netapp-config.yaml
environments/manila-vmax-config.yaml
environments/manila-cephfsnative-config.yaml
environments/manila-cephfsganesha-config.yaml
environments/manila-unity-config.yaml
environments/manila-vnx-config.yaml
```

환경 파일 사용자 지정 및 배포에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- *공유 파일 시스템 서비스의 NFS 백엔드 가이드를 통해 CephFS* 에서 업데이트된 환경 배포
- 공유 파일 시스템 서비스 *용 NetApp 백엔드 가이드의 NetApp 백엔드와 함께 공유 파일 시스템 서비스*배포
- 공유 파일 시스템 서비스를 *위한 CephFS 백엔드의 CephFS 백엔드 가이드를 사용하여 공유 파일 시스템 서비스*배포

5.4. RED HAT 레지스트리를 원격 레지스트리 소스로 사용

Red Hat은 **registry.redhat.io** 에서 오버클라우드 컨테이너 이미지를 호스팅합니다. 레지스트리가 이미 구성되어 있고 필요한 모든 것은 가져올 이미지의 URL과 네임스페이스이기 때문에 원격 레지스트리에서 이미지를 가져오는 것이 가장 간단한 방법입니다. 그러나 오버클라우드 생성 중에 오버클라우드 노드는 모두 원격 리포지토리에서 이미지를 가져와 외부 연결을 끊을 수 있습니다. 따라서 프로덕션 환경에는 이 방법을 사용하지 않는 것이 좋습니다. 프로덕션 환경의 경우 다음 방법 중 하나를 사용합니다.

- 로컬 레지스트리 설정
- Red Hat Satellite 6에서 이미지 호스트

절차

1. 오버클라우드 배포의 **registry.redhat.io** 에서 이미지를 직접 가져오려면 이미지 매개변수를 지정하려면 환경 파일이 필요합니다. 다음 명령을 실행하여 컨테이너 이미지 환경 파일을 생성합니다.

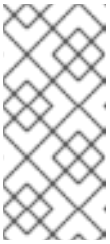
```
(undercloud) $ sudo openstack overcloud container image prepare \
  --namespace=registry.redhat.io/rhosp13 \
  --prefix=openstack- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

   - 옵션 서비스에 대한 환경 파일을 포함하려면 **-e** 옵션을 사용합니다.
   - 사용자 지정 역할 파일을 포함하려면 **-r** 옵션을 사용합니다.
   - Ceph Storage를 사용하는 경우 추가 매개변수를 포함하여 Ceph Storage 컨테이너 이미지 위치 **--set ceph_namespace,--set ceph_image,--set ceph_tag**.

2. **overcloud_images.yaml** 파일을 수정하고 다음 매개변수를 포함하여 배포 중에 **registry.redhat.io** 로 인증합니다.

```
ContainerImageRegistryLogin: true
ContainerImageRegistryCredentials:
  registry.redhat.io:
    <USERNAME>: <PASSWORD>
```

- **<USERNAME>** 및 **<PASSWORD>** 를 **registry.redhat.io** 의 자격 증명으로 바꿉니다. **overcloud_images.yaml** 파일에는 언더클라우드의 이미지 위치가 포함되어 있습니다. 배포와 함께 이 파일을 포함합니다.



참고

openstack overcloud deploy 명령을 실행하기 전에 원격 레지스트리에 로그인해야 합니다.

```
(undercloud) $ sudo docker login registry.redhat.io
```

레지스트리 구성이 준비되었습니다.

5.5. 언더클라우드를 로컬 레지스트리로 사용

오버클라우드 컨테이너 이미지를 저장하도록 언더클라우드에서 로컬 레지스트리를 구성할 수 있습니다.

director를 사용하여 **registry.redhat.io** 에서 각 이미지를 가져와서 언더클라우드에서 실행되는 **docker-distribution** 레지스트리로 각 이미지를 푸시할 수 있습니다. director를 사용하여 오버클라우드 생성 프로세스 중에 노드가 언더클라우드 **docker-distribution** 레지스트리에서 관련 이미지를 가져옵니다.

이렇게 하면 내부 네트워크 내에 컨테이너 이미지에 대한 네트워크 트래픽이 유지되므로 외부 네트워크 연결을 제한하지 않고 배포 프로세스의 속도를 높일 수 있습니다.

절차

1. 로컬 언더클라우드 레지스트리의 주소를 찾습니다. 주소는 다음 패턴을 사용합니다.

```
<REGISTRY_IP_ADDRESS>:8787
```

   이전에 **undercloud.conf** 파일의 **local_ip** 매개변수로 설정한 언더클라우드의 IP 주소를 사용합니다. 아래 명령의 경우 주소는 **192.168.24.1:8787** 이라고 합니다.

2. **registry.redhat.io** 에 로그인합니다.

```
(undercloud) $ docker login registry.redhat.io --username $RH_USER --password
$RH_PASSWD
```

3. 이미지를 로컬 레지스트리에 업로드하는 템플릿과 해당 이미지를 참조할 환경 파일을 생성합니다.

```
(undercloud) $ openstack overcloud container image prepare \
  --namespace=registry.redhat.io/rhosp13 \
  --push-destination=192.168.24.1:8787 \
  --prefix=openstack- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml \
  --output-images-file /home/stack/local_registry_images.yaml
```

-

- 옵션 서비스에 대한 환경 파일을 포함하려면 **-e** 옵션을 사용합니다.
- 사용자 지정 역할 파일을 포함하려면 **-r** 옵션을 사용합니다.
- Ceph Storage를 사용하는 경우 추가 매개변수를 포함하여 Ceph Storage 컨테이너 이미지 위치 **--set ceph_namespace,--set ceph_image,--set ceph_tag**.

4. 다음 두 파일이 생성되었는지 확인합니다.
   - **local_registry_images.yaml**: 원격 소스의 컨테이너 이미지 정보가 포함됩니다. 이 파일을 사용하여 Red Hat Container Registry(**registry.redhat.io**)에서 언더클라우드로 이미지를 가져옵니다.
   - undercloud의 최종 이미지 위치가 포함된 **overcloud_images.yaml**. 배포와 함께 이 파일을 포함합니다.
5. 원격 레지스트리에서 컨테이너 이미지를 가져와서 언더클라우드 레지스트리로 푸시합니다.

```
(undercloud) $ openstack overcloud container image upload \
 --config-file  /home/stack/local_registry_images.yaml \
 --verbose
```

네트워크 및 언더클라우드 디스크에 따라 필요한 이미지를 가져오는 데 시간이 다소 걸릴 수 있습니다.

참고

컨테이너 이미지는 약 10GB의 디스크 공간을 사용합니다.

6. 이제 이미지가 언더클라우드의 **docker-distribution** 레지스트리에 저장됩니다. 언더클라우드의 **docker-distribution** 레지스트리에서 이미지 목록을 보려면 다음 명령을 실행합니다.

```
(undercloud) $  curl http://192.168.24.1:8787/v2/_catalog | jq .repositories[]
```

참고

_catalog 리소스 자체는 100개의 이미지만 표시합니다. 더 많은 이미지를 표시하려면 더 많은 수의 이미지를 표시하려면 **_catalog** 리소스와 함께 **?n=<interger>** 쿼리 문자열을 사용합니다.

```
(undercloud) $  curl http://192.168.24.1:8787/v2/_catalog?n=150 | jq
.repositories[]
```

특정 이미지의 태그 목록을 보려면 **skopeo** 명령을 사용합니다.

```
(undercloud) $ curl -s http://192.168.24.1:8787/v2/rhosp13/openstack-keystone/tags/list | jq
.tags
```

태그된 이미지를 확인하려면 **skopeo** 명령을 사용합니다.

```
(undercloud) $ skopeo inspect --tls-verify=false
docker://192.168.24.1:8787/rhosp13/openstack-keystone:13.0-44
```

레지스트리 구성이 준비되었습니다.

5.6. SATELLITE 서버를 레지스트리로 사용

Red Hat Satellite 6는 레지스트리 동기화 기능을 제공합니다. 이를 통해 여러 이미지를 Satellite 서버로 가져와 애플리케이션 라이프사이클의 일부로 관리할 수 있습니다. Satellite는 다른 컨테이너 활성화 시스템이 사용할 레지스트리 역할도 합니다. 컨테이너 이미지 관리 방법에 대한 자세한 내용은 *Red Hat Satellite 6 Content Management Guide*의 "Managing Container Images"를 참조하십시오.

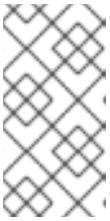
다음 절차의 예제에서는 Red Hat Satellite 6용 **hammer** 명령행 툴과 **ACME**라는 조직을 사용합니다. 이 조직을 실제로 사용하는 Satellite 6 조직으로 대체하십시오.

절차

1. 로컬 레지스트리로 이미지를 가져오는 템플릿을 생성합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud container image prepare \
  --namespace=rhosp13 \
  --prefix=openstack- \
  --output-images-file /home/stack/satellite_images
```

   - 옵션 서비스에 대한 환경 파일을 포함하려면 **-e** 옵션을 사용합니다.
   - 사용자 지정 역할 파일을 포함하려면 **-r** 옵션을 사용합니다.
   - Ceph Storage를 사용하는 경우 추가 매개변수를 포함하여 Ceph Storage 컨테이너 이미지 위치 **--set ceph_namespace,--set ceph_image,--set ceph_tag**.



참고

이 버전의 **openstack overcloud 컨테이너 image prepare** 명령은 **registry.redhat.io** 의 레지스트리를 대상으로 이미지 목록을 생성합니다. 이후 단계에서 사용된 **openstack overcloud 컨테이너 이미지 prepare** 명령과 다른 값을 사용합니다.

2. 이렇게 하면 컨테이너 이미지 정보와 함께 **satellite_images** 라는 파일이 생성됩니다. 이 파일을 사용하여 컨테이너 이미지를 Satellite 6 서버에 동기화합니다.

3. **satellite_images** 파일에서 YAML 관련 정보를 제거하고 이미지 목록만 포함된 플랫 파일로 변환합니다. 다음 **sed** 명령은 이를 수행합니다.

```
(undercloud) $ awk -F ':' '{if (NR!=1) {gsub("[[:space:]]", ""); print $2}}' ~/satellite_images > ~/satellite_images_names
```

   이는 Satellite 서버에 가져온 이미지 목록을 제공합니다.

4. **satellite_images_names** 파일을 Satellite 6 **hammer** 툴이 포함된 시스템으로 복사합니다. 또는 *Hammer CLI 가이드*의 지침에 따라 **hammer** 툴을 언더클라우드에 설치합니다.

5. 다음 **hammer** 명령을 실행하여 Satellite 조직에 새 제품(**OSP13 Containers**)을 생성합니다.

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP13 Containers"
```

이 사용자 지정 제품에 이미지를 저장합니다.

6. 제품에 기본 컨테이너 이미지를 추가합니다.

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP13 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhosp13/openstack-base \
  --name base
```

7. **satellite_images** 파일에서 오버클라우드 컨테이너 이미지를 추가합니다.

```
$ while read IMAGE; do \
  IMAGENAME=$(echo $IMAGE | cut -d"/" -f2 | sed "s/openstack-//g" | sed "s/:.*//g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP13 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE \
  --name $IMAGENAME ; done < satellite_images_names
```

8. 컨테이너 이미지를 동기화합니다.

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP13 Containers"
```

Satellite 서버가 동기화를 완료할 때까지 기다립니다.

참고

설정에 따라 **hammer**에서 Satellite 서버 사용자 이름과 암호가 필요할 수 있습니다. **hammer**를 구성한 후 구성 파일을 사용하여 자동으로 로그인할 수 있습니다. *Hammer CLI Guide* 의 "Authentication" 섹션을 참조하십시오.

9. Satellite 6 서버에서 콘텐츠 뷰를 사용하는 경우 새 콘텐츠 뷰 버전을 생성하여 이미지를 통합합니다.

10. **기본** 이미지에 사용 가능한 태그를 확인합니다.

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --product "OSP13 Containers"
```

그러면 OpenStack Platform 컨테이너 이미지에 대한 태그가 표시됩니다.

11. 언더클라우드로 돌아가서 Satellite 서버의 이미지에 대한 환경 파일을 생성합니다. 다음은 환경 파일을 생성하는 예제 명령입니다.

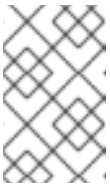
```
(undercloud) $ openstack overcloud container image prepare \
  --namespace=satellite6.example.com:5000 \
  --prefix=acme-osp13_containers- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

참고

이 버전의 **openstack overcloud 컨테이너 image prepare** 명령은 Satellite 서버를 대상으로 합니다. 이전 단계에서 사용한 **openstack overcloud 컨테이너 이미지 prepare** 명령과 다른 값을 사용합니다.

이 명령을 실행하는 경우 다음 데이터를 포함합니다.

- **--namespace** - Satellite 서버에 있는 레지스트리의 URL 및 포트입니다. Red Hat Satellite의 레지스트리 포트는 5000입니다. 예를 들면 **--namespace=satellite6.example.com:5000** 입니다.



참고

Red Hat Satellite 버전 6.10을 사용하는 경우 포트를 지정할 필요가 없습니다. 기본 포트 **443** 이 사용됩니다. 자세한 내용은 " RHOSP13 배포를 Red Hat Satellite 6.10에 어떻게 적용할 수 있습니까?"에서 참조하십시오.

- **--prefix=** - 접두사는 소문자를 사용하고 밑줄로 이루어진 공백을 대체하는 라벨에 대한 Satellite 6 규칙을 기반으로 합니다. 접두사는 콘텐츠 뷰 사용 여부에 따라 달라집니다.
  - 콘텐츠 뷰를 사용하는 경우 구조는 **[org]-[environment]-[content view]-[product]-**입니다. 예를 들면 **acme-production-myosp13-osp13_containers-** 입니다.
  - 콘텐츠 뷰를 사용하지 않는 경우 구조는 **[org]-[product]-**입니다. 예: **acme-osp13_containers-**.
- **--tag-from-label {version}-{release}** - 각 이미지의 최신 태그를 식별합니다.
- **-e** - 선택적 서비스에 대한 환경 파일을 포함합니다.
- **-** R - 사용자 지정 역할 파일을 포함합니다.
- **--set ceph_namespace**,**--set ceph_image**,**--set ceph_tag** - Ceph Storage를 사용하는 경우 추가 매개변수를 포함하여 Ceph Storage 컨테이너 이미지 위치를 정의합니다. 이제 **ceph_image**에는 Satellite별 접두사가 포함됩니다. 이 접두사는 **--prefix** 옵션과 동일한 값입니다. 예를 들면 다음과 같습니다.

```
--set ceph_image=acme-osp13_containers-rhceph-3-rhel7
```

이렇게 하면 오버클라우드에서 Satellite 이름 지정 규칙을 사용하여 Ceph 컨테이너 이미지를 사용할 수 있습니다.

12. **overcloud_images.yaml** 파일에는 Satellite 서버의 이미지 위치가 포함되어 있습니다. 배포와 함께 이 파일을 포함합니다.

레지스트리 구성이 준비되었습니다.

5.7. 다음 단계

이제 컨테이너 이미지 소스 목록이 포함된 **overcloud_images.yaml** 환경 파일이 있습니다. 향후 모든 배포 작업과 함께 이 파일을 포함합니다.

6장. CLI 툴로 기본 오버클라우드 구성

이 장에서는 CLI 툴을 사용하는 OpenStack Platform 환경의 기본 구성 단계를 설명합니다. 기본 설정을 사용하는 오버클라우드에는 사용자 지정 기능이 없습니다. 하지만 Advanced Overcloud Customization 가이드의 지침에 따라 이 기본 오버클라우드에 고급 옵션을 추가하고 사양에 맞게 사용자 지정할 수 있습니다.

이 장의 예에서는 모든 노드가 전원 관리에 IPMI를 사용하는 베어 메탈 시스템입니다. 지원되는 추가 전원 관리 유형 및 옵션은 부록 B. *전원 관리 드라이버* 을 참조하십시오.

워크플로

1. 노드 정의 템플릿을 생성하고 director에 빈 노드를 등록합니다.
2. 모든 노드의 하드웨어를 검사합니다.
3. 노드를 역할에 태그합니다.
4. 추가 노드 속성을 정의합니다.

요구 사항

- 에서 생성한 director 노드 4장. *언더클라우드 설치*
- 노드에 사용할 베어 메탈 시스템 세트. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다(Overcloud 역할에 대한 내용은 3.1절. "노드 배포 역할 계획" 참조). 또한 이러한 머신은 각 노드 유형에 설정된 요구 사항을 준수해야 합니다. 이러한 요구 사항은 2.4절. "오버클라우드 요구 사항" 에서 참조하십시오. 이러한 노드에는 운영 체제가 필요하지 않습니다. director는 Red Hat Enterprise Linux 7 이미지를 각 노드에 복사합니다.
- 기본 VLAN으로 구성된 프로비저닝 네트워크에 대한 하나의 네트워크 연결입니다. 모든 노드는 이 네트워크에 연결되어야 하며 2.3절. "네트워킹 요구 사항" 에 설정된 요구 사항을 준수해야 합니다. 이 장의 예제에서는 다음 IP 주소가 할당된 프로비저닝 서브넷으로 192.168.24.0/24를 사용합니다.

  표 6.1. 프로비저닝 네트워크 IP 할당

노드 이름	IP 주소	MAC 주소	IPMI IP 주소
Director	192.168.24.1	A:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a:a	필수 없음
컨트롤러	DHCP 정의	bb:bb:bb:bb:bb:bb	192.168.24.205
Compute	DHCP 정의	cc:cc:cc:cc:cc:cc	192.168.24.206

- 다른 모든 네트워크 유형은 OpenStack 서비스에 프로비저닝 네트워크를 사용합니다. 하지만 다른 네트워크 트래픽 유형에 대해 추가 네트워크를 생성할 수 있습니다.
- 컨테이너 이미지의 소스입니다. 컨테이너 이미지 소스가 포함된 환경 파일을 생성하는 방법에 대한 자세한 내용은 5장. *컨테이너 이미지 소스 구성* 을 참조하십시오.

6.1. 오버클라우드에 노드 등록

director에는 수동으로 만든 노드 정의 템플릿이 있어야 합니다. 이 파일(**instackenv.json**)은 JSON 형식을 사용하고 노드의 하드웨어 및 전원 관리 세부 정보를 포함합니다. 예를 들어 두 노드를 등록하기 위한 템플릿은 다음과 같습니다.

```
{
    "nodes":[
        {
            "mac":[
                "bb:bb:bb:bb:bb:bb"
            ],
            "name":"node01",
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"ipmi",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.168.24.205"
        },
        {
            "mac":[
                "cc:cc:cc:cc:cc:cc"
            ],
            "name":"node02",
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"ipmi",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.168.24.206"
        }
    ]
}
```

이 템플릿에서는 다음 속성을 사용합니다.

name

노드의 논리 이름입니다.

pm_type

사용할 전원 관리 드라이버입니다. 이 예에서는 전원 관리를 위한 기본 드라이버인 IPMI 드라이버(**ipmi**)를 사용합니다.

참고

IPMI는 지원되는 기본 전원 관리 드라이버입니다. 지원되는 추가 전원 관리 유형 및 옵션은 부록 B. *전원 관리 드라이버* 을 참조하십시오. 이러한 전원 관리 드라이버가 예상대로 작동하지 않는 경우 IPMI를 전원 관리에 사용합니다.

pm_user; pm_password

IPMI 사용자 이름 및 암호입니다. 이러한 속성은 IPMI 및 Redfish에 대해 선택 사항이며 iLO 및 iDRAC에는 필수입니다.

pm_addr

IPMI 장치의 IP 주소입니다.

pm_port

(선택 사항) 특정 IPMI 장치에 액세스하는 포트입니다.

mac

(선택 사항) 노드에 있는 네트워크 인터페이스의 MAC 주소 목록입니다. 각 시스템의 프로비저닝 NIC에는 MAC 주소만 사용합니다.

cpu

(선택 사항) 노드에 있는 CPU 수입니다.

memory

(선택 사항) 메모리 크기(MB)입니다.

disk

(선택 사항) 하드 디스크의 크기(GB)입니다.

arch

(선택 사항) 시스템 아키텍처입니다.

중요

다중 아키텍처 클라우드를 빌드하는 경우 **x86_64** 및 **ppc64le** 아키텍처를 사용하여 노드를 구분하려면 **arch** 키가 필요합니다.

템플릿을 생성한 후 다음 명령을 실행하여 포맷과 구문을 확인합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud node import --validate-only ~/instackenv.json
```

stack 사용자의 홈 디렉터리(**/home/stack/instackenv.json**)에 파일을 저장하고 다음 명령을 실행하여 템플릿을 director로 가져옵니다.

```
(undercloud) $ openstack overcloud node import ~/instackenv.json
```

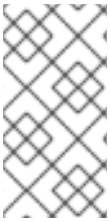
이렇게 하면 템플릿을 가져와서 템플릿의 각 노드가 director에 등록됩니다.

노드 등록 및 구성이 완료되면 CLI에서 이러한 노드 목록을 확인합니다.

```
(undercloud) $ openstack baremetal node list
```

6.2. 노드의 하드웨어 검사

director는 각 노드에서 인트로스펙션 프로세스를 실행할 수 있습니다. 이 프로세스에서는 각 노드가 PXE를 통해 인트로스펙션 에이전트를 부팅합니다. 이 에이전트는 노드에서 하드웨어 데이터를 수집하고 director로 다시 보냅니다. 그러면 director가 이 인트로스펙션 데이터를 director에서 실행 중인 OpenStack Object Storage(swift) 서비스에 저장합니다. director는 프로필 태그, 벤치마킹, 수동 루트 디스크 할당과 같은 다양한 목적으로 하드웨어 정보를 사용합니다.



참고

정책 파일을 생성하여 인트로스펙션 직후 노드를 프로필에 자동으로 태그할 수도 있습니다. 정책 파일 생성 및 인트로스펙션 프로세스에 해당 파일을 포함하는 방법에 대한 자세한 내용은 부록 E. *프로파일 자동 태그* 을 참조하십시오. 또는 6.5절. "프로필에 노드 태그" 의 지침에 따라 수동으로 노드를 프로필에 태그할 수 있습니다.

다음 명령을 실행하여 각 노드의 하드웨어 속성을 확인합니다.

```
(undercloud) $ openstack overcloud node introspect --all-manageable --provide
```

- **--all-manageable** 옵션은 관리 상태의 노드만 인트로스펙션합니다. 이 예제에서는 모든 것입니다.
- **--provide** 옵션은 인트로스펙션 이후 모든 노드를 **available** 상태로 리셋합니다.

다른 터미널 창에서 다음 명령을 사용하여 인트로스펙션 진행 상황을 모니터링합니다.

```
(undercloud) $ sudo journalctl -l -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq
-u openstack-ironic-conductor -f
```

중요

이 프로세스가 완료되었는지 확인합니다. 베어 메탈 노드의 경우 이 프로세스는 일반적으로 15분 정도 걸립니다.

인트로스펙션이 완료되면 모든 노드가 **available** 상태로 변경됩니다.

노드에 대한 인트로스펙션 정보를 보려면 다음 명령을 실행합니다.

```
(undercloud) $ openstack baremetal introspection data save <UUID> | jq .
```

<UUID> 를 인트로스펙션 정보를 검색할 노드의 UUID로 바꿉니다.

개별적으로 노드 인트로스펙션 수행

사용 가능한 노드에서 단일 인트로스펙션을 수 행하려면 노드를 관리 모드로 설정하고 인트로스펙션을 수행합니다.

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

인트로스펙션이 완료되면 노드가 **available** 상태로 변경됩니다.

초기 인트로스펙션 이후 노드 인트로스펙션 수행

초기 인트로스펙션 이후 모든 노드는 **--provide** 옵션으로 인해 **available** 상태가 됩니다. 초기 인트로스펙션 이후 모든 노드에서 인트로스펙션을 수행하려면 모든 노드를 **manageable** 상태로 설정하고 bulk introspection 명령을 실행합니다.

```
(undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do openstack
baremetal node manage $node ; done
(undercloud) $ openstack overcloud node introspect --all-manageable --provide
```

인트로스펙션이 완료되면 모든 노드가 **available** 상태로 변경됩니다.

인터페이스 정보에 대한 네트워크 인트로스펙션 수행

네트워크 인트로스펙션은 네트워크 스위치에서 LLDP(link layer discovery protocol) 데이터를 검색합니다. 다음 명령은 노드의 모든 인터페이스에 대한 LLDP 정보 서브셋이나 특정 노드 및 인터페이스에 대한 전체 정보를 표시합니다. 이 정보는 문제 해결에 유용할 수 있습니다. director는 기본적으로 LLDP 데이터 수집을 활성화합니다.

노드에서 인터페이스 목록을 가져오려면 다음을 수행합니다.

```
(undercloud) $ openstack baremetal introspection interface list [NODE UUID]
```

예를 들면 다음과 같습니다.

```
(undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-
79f8b86c7cd9
+-----------+-------------------+------------------------+-------------------+----------------+
| Interface | MAC Address       | Switch Port VLAN IDs   | Switch Chassis ID | Switch Port ID |
+-----------+-------------------+------------------------+-------------------+----------------+
| p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510            |
| p2p1      | 00:0a:f7:79:93:18 | [101]                  | 64:64:9b:31:12:00 | 507            |
| em1       | c8:1f:66:c7:e8:2f | [162]                  | 08:81:f4:a6:b3:80 | 515            |
| em2       | c8:1f:66:c7:e8:30 | [182, 183]             | 08:81:f4:a6:b3:80 | 559            |
+-----------+-------------------+------------------------+-------------------+----------------+
```

인터페이스 데이터 및 스위치 포트 정보를 보려면 다음을 수행합니다.

```
(undercloud) $ openstack baremetal introspection interface show [NODE UUID] [INTERFACE]
```

예를 들면 다음과 같습니다.

```
(undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-907d-
79f8b86c7cd9 p2p1
+--------------------------------------+------------------------------------------------------------------------------------
--------------------------------------+
| Field                                | Value
|
+--------------------------------------+------------------------------------------------------------------------------------
--------------------------------------+
| interface                            | p2p1
|
| mac                                  | 00:0a:f7:79:93:18
|
| node_ident                           | c89397b7-a326-41a0-907d-79f8b86c7cd9
|
| switch_capabilities_enabled          | [u'Bridge', u'Router']
|
| switch_capabilities_support          | [u'Bridge', u'Router']
|
| switch_chassis_id                    | 64:64:9b:31:12:00
|
| switch_port_autonegotiation_enabled  | True
|
| switch_port_autonegotiation_support  | True
```

```
|
| switch_port_description              | ge-0/0/2.0
|
| switch_port_id                       | 507
|
| switch_port_link_aggregation_enabled | False
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type                 | Unknown
|
| switch_port_mtu                      | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-TX hdx',
u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids        | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id         | 101
|
| switch_port_vlan_ids                 | [101]
|
| switch_port_vlans                    | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities           | None
|
| switch_system_name                   | rhos-compute-node-sw1
|
+--------------------------------------+-----------------------------------------------------------------------------
--------------------------------------+
```

하드웨어 인트로스펙션 세부 정보 검색

Bare Metal 서비스 하드웨어 검사 추가 기능(**inspection_extras**)은 기본적으로 활성화되어 하드웨어 세부 정보를 검색합니다. 이러한 하드웨어 세부 정보를 사용하여 오버클라우드를 구성할 수 있습니다. **undercloud.conf** 파일의 **inspection_extras** 매개변수에 대한 자세한 내용은 Director *설치 및 사용 가이드의 Director* 설정을 참조하십시오.

예를 들어 **numa_topology** 수집기는 이러한 하드웨어 검사의 추가 기능의 일부이며 각 NUMA 노드에 대해 다음 정보를 포함합니다.

- RAM(KB)
- 물리적 CPU 코어 및 시블링 스레드
- NUMA 노드와 연결된 NIC

openstack baremetal introspection data save _UUID_ | jq .numa_topology 명령을 사용하여 베어 메탈 노드의 *UUID* 로 이 정보를 검색합니다.

다음 예제는 베어 메탈 노드에 대해 검색된 NUMA 정보를 보여줍니다.

```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
        0,
        16
      ],
      "numa_node": 0
    },
    {
      "cpu": 5,
      "thread_siblings": [
        13,
        29
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        15,
        31
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        7,
        23
      ],
      "numa_node": 0
    },
    {
      "cpu": 1,
      "thread_siblings": [
        9,
```

```
      25
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      6,
      22
    ],
    "numa_node": 0
  },
  {
    "cpu": 3,
    "thread_siblings": [
      11,
      27
    ],
    "numa_node": 1
  },
  {
    "cpu": 5,
    "thread_siblings": [
      5,
      21
    ],
    "numa_node": 0
  },
  {
    "cpu": 4,
    "thread_siblings": [
      12,
      28
    ],
    "numa_node": 1
  },
  {
    "cpu": 4,
    "thread_siblings": [
      4,
      20
    ],
    "numa_node": 0
  },
  {
    "cpu": 0,
    "thread_siblings": [
      8,
      24
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      14,
```

```
        30
      ],
      "numa_node": 1
    },
    {
      "cpu": 3,
      "thread_siblings": [
        3,
        19
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        2,
        18
      ],
      "numa_node": 0
    }
  ],
  "ram": [
    {
      "size_kb": 66980172,
      "numa_node": 0
    },
    {
      "size_kb": 67108864,
      "numa_node": 1
    }
  ],
  "nics": [
    {
      "name": "ens3f1",
      "numa_node": 1
    },
    {
      "name": "ens3f0",
      "numa_node": 1
    },
    {
      "name": "ens2f0",
      "numa_node": 0
    },
    {
      "name": "ens2f1",
      "numa_node": 0
    },
    {
      "name": "ens1f1",
      "numa_node": 0
    },
    {
      "name": "ens1f0",
      "numa_node": 0
    },
```

```
    {
      "name": "eno4",
      "numa_node": 0
    },
    {
      "name": "eno1",
      "numa_node": 0
    },
    {
      "name": "eno3",
      "numa_node": 0
    },
    {
      "name": "eno2",
      "numa_node": 0
    }
  ]
}
```

6.3. 베어 메탈 노드 자동 검색

instackenv.json 파일을 먼저 생성하지 않고 *auto-discovery* 를 사용하여 언더클라우드 노드를 등록하고 해당 메타데이터를 생성할 수 있습니다. 이러한 개선 사항을 통해 초기 노드 정보를 수집하는 데 소요되는 시간을 줄일 수 있습니다. 예를 들어 IPMI IP 주소를 분석하고 나중에 **instackenv.json** 을 생성할 필요가 없습니다.

요구 사항

- 모든 오버클라우드 노드는 IPMI를 통해 director에 액세스할 수 있도록 BMC를 구성해야 합니다.
- 모든 오버클라우드 노드는 언더클라우드 컨트롤 플레인 네트워크에 연결된 NIC에서 PXE 부팅되도록 구성해야 합니다.

자동 검색 활성화

1. 베어 메탈 자동 검색은 **undercloud.conf** 에서 활성화됩니다.

```
enable_node_discovery = True
discovery_default_driver = ipmi
```

   - **enable_node_discovery** - 활성화하면, PXE를 사용하여 인트로스펙션 램디스크를 부팅하는 모든 노드가 ironic에 등록됩니다.
   - **discovery_default_driver** - 검색된 노드에 사용할 드라이버를 설정합니다. 예를 들어 **ipmi**입니다.

2. ironic에 IPMI 인증서를 추가합니다.

   a. **ipmi-credentials.json**이라는 파일에 IPMI 인증서를 추가합니다. 이 예제의 사용자 이름과 암호 값을 사용자 환경에 맞게 변경해야 합니다.

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
```

```
            {"op": "eq", "field": "data://auto_discovered", "value": true}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver_info/ipmi_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/ipmi_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/ipmi_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    }
]
```

3. IPMI 인증서 파일을 ironic으로 가져옵니다.

```
$ openstack baremetal introspection rule import ipmi-credentials.json
```

자동 검색 테스트

1. 필요한 노드의 전원을 켭니다.
2. **openstack baremetal node list** 를 실행합니다. 새 노드가 **enrolled** 상태로 표시됩니다.

```
$ openstack baremetal node list
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             |
False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             |
False       |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
```

3. 각 노드에 리소스 클래스를 설정합니다.

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node set $NODE --resource-class baremetal ; done
```

4. 각 노드에 커널 및 램디스크를 설정합니다.

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. 모든 노드를 사용 가능한 상태로 설정합니다.

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node provide $NODE ; done
```

규칙을 사용하여 다른 벤더 하드웨어 검색 방법

여러 가지가 혼합된 하드웨어 환경의 경우 인트로스펙션 규정을 사용하여 인증서 및 원격 관리 인증서를 할당할 수 있습니다. 예를 들어 별도의 검색 규칙으로 DRAC를 사용하는 Dell 노드를 처리할 수 있습니다.

1. 다음 내용과 함께 **dell-drac-rules.json** 이라는 파일을 생성합니다. 이 예제의 사용자 이름과 암호 값을 사용자 환경에 맞게 변경해야 합니다.

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true},
      {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
       "value": "Dell Inc."}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  },
  {
    "description": "Set the vendor driver for Dell hardware",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true},
      {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
       "value": "Dell Inc."}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver", "value": "idrac"},
      {"action": "set-attribute", "path": "driver_info/drac_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/drac_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/drac_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  }
]
```

2. 규칙을 ironic으로 가져옵니다.

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

6.4. 아키텍처별 역할 생성

다중 아키텍처 클라우드를 빌드하는 경우 모든 아키텍처별 역할을 **roles_data.yaml** 에 추가해야 합니다. 다음은 기본 역할과 함께 **ComputePPC64LE** 역할을 포함하는 예입니다. Creating a Custom Role File 섹션에는 역할에 관한 정보가 있습니다.

```
openstack overcloud roles generate \
  --roles-path /usr/share/openstack-tripleo-heat-templates/roles -o ~/templates/roles_data.yaml \
  Controller Compute ComputePPC64LE BlockStorage ObjectStorage CephStorage
```

6.5. 프로필에 노드 태그

각 노드의 하드웨어 등록 및 검사 후 특정 프로필에 태그를 지정합니다. 이러한 프로필 태그는 노드와 플레이버가 배포 역할에 할당됩니다. 다음 예제에서는 컨트롤러 노드의 역할, 플레이버, 프로필 및 노드 간의 관계를 보여줍니다.

유형	설명
Role	**Controller** 역할은 컨트롤러 노드 구성 방법을 정의합니다.
플레이버	**control** 플레이버는 노드에서 컨트롤러로 사용할 하드웨어 프로필을 정의합니다. director에서 사용할 노드를 결정할 수 있도록 이를 **Controller** 역할에 할당합니다.
프로필	**control** 프로필은 **control** 플레이버에 적용하는 태그입니다. 이 프로필은 플레이버에 속한 노드를 정의합니다.
노드	또한 **control** 프로필 태그를 개별 노드에 적용하면 **control** 플레이버에 그룹화되며, 결과적으로 director에서는 **Controller** 역할을 사용하여 이러한 개별 노드를 설정합니다.

기본 프로필 플레이버 **compute**, **control**, **swift-storage**, **ceph-storage**, **block-storage**는 언더클라우드 설치 중에 생성되며, 대부분의 환경에서 변경 없이 사용할 수 있습니다.

참고

다수의 노드의 경우 자동 프로필 태깅을 사용합니다. 자세한 내용은 부록 E. *프로파일 자동 태그*를 참조하십시오.

노드를 특정 프로필에 태그하려면 **profile** 옵션을 각 노드의 **properties/capabilities** 매개변수에 추가합니다. 예를 들어 각각 컨트롤러 및 컴퓨팅 프로필을 사용하도록 노드를 태그하려면 다음 명령을 사용합니다.

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:compute,boot_option:local' 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13
(undercloud) $ openstack baremetal node set --property capabilities='profile:control,boot_option:local'
1a4e30da-b6dc-499d-ba87-0bd8a3819bc0
```

profile:compute 및 **profile:control** 옵션을 추가하면 두 개의 노드가 각각 해당하는 프로필에 태그됩니다.

이러한 명령은 각 노드를 부팅하는 방법을 정의하는 **boot_option:local** 매개변수도 설정합니다. 하드웨어에 따라 기본 BIOS 모드 대신 UEFI를 사용하여 노드를 부팅하도록 **boot_mode** 매개변수를 **uefi** 에 추가해야 할 수도 있습니다. 자세한 내용은 D.2절. "UEFI 부팅 모드"의 내용을 참조하십시오.

노드 태그를 완료한 후 할당된 프로필 또는 가능한 프로필을 확인합니다.

```
(undercloud) $ openstack overcloud profiles list
```

사용자 정의 역할 프로필

사용자 지정 역할을 사용하는 경우 이러한 새 역할을 수용할 수 있도록 추가 플레이버 및 프로필을 만들어야 할 수 있습니다. 예를 들어 Networker 역할에 대한 새 플레이버를 생성하려면 다음 명령을 실행합니다.

```
(undercloud) $ openstack flavor create --id auto --ram 4096 --disk 40 --vcpus 1 networker
(undercloud) $ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property "capabilities:profile"="networker" networker
```

이 새 프로필로 노드를 할당합니다.

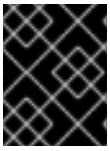
```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:networker,boot_option:local' dad05b82-0c74-40bf-9d12-193184bfc72d
```

6.6. ROOT 디스크 정의

여러 디스크가 있는 노드의 경우 director가 프로비저닝 중에 root 디스크를 식별해야 합니다. 예를 들어 대부분의 Ceph Storage 노드는 여러 디스크를 사용합니다. 기본적으로 director는 프로비저닝 프로세스 중에 오버클라우드 이미지를 root 디스크에 씁니다.

director가 root 디스크를 쉽게 식별할 수 있도록 다음과 같은 여러 속성을 정의할 수 있습니다.

- **모델** (문자열): 장치 식별자.
- **vendor** (문자열): 장치 벤더.
- **serial** (문자열): 디스크 일련 번호입니다.
- **hctl** (문자열): host:Channel:Target:Lun(SCSI용)
- **크기** (정수): 장치 크기(GB)입니다.
- **wwn** (문자열): 고유한 스토리지 식별자입니다.
- **wwn_with_extension** (문자열): 공급업체 확장이 추가된 고유한 스토리지 식별자입니다.
- **wwn_vendor_extension** (문자열): 고유한 벤더 스토리지 식별자입니다.
- **rotational** (Boolean): 회전 장치(HDD)의 경우 true이고, 그렇지 않으면 false(SSD)입니다.
- **이름** (문자열): 장치 이름(예: /dev/sdb1).
- **by_path** (문자열): 장치의 고유한 PCI 경로입니다. 장치의 UUID를 사용하지 않으려면 이 속성을 사용합니다.



중요

name 속성은 영구적인 이름이 있는 장치에만 사용합니다. 노드가 부팅될 때 값이 변경될 수 있으므로 **name** 을 사용하여 다른 장치에 대해 root 디스크를 설정하지 마십시오.

일련번호를 사용하여 root 장치를 지정하려면 다음 단계를 완료합니다.

절차

1. 각 노드의 하드웨어 인트로스펙션에서 디스크 정보를 확인합니다. 다음 명령을 실행하여 노드의 디스크 정보를 표시합니다.

```
(undercloud) $ openstack baremetal introspection data save 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 | jq ".inventory.disks"
```

예를 들어 노드 1개의 데이터에서 디스크 3개가 표시될 수 있습니다.

```
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
    "wwn_vendor_extension": "0x1ea4e31e121cfb45",
    "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
```

2. 노드 정의의 **root_device** 매개 변수로 변경합니다. 다음 예제에서는 root 장치를 일련 번호가 **61866da04f380d001ea4e13c12e36ad6** 인 disk 2로 설정하는 방법을 보여줍니다.

```
(undercloud) $ openstack baremetal node set --property root_device='{"serial":
"61866da04f380d001ea4e13c12e36ad6"}' 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0
```

참고

각 노드의 BIOS를 설정하여 선택한 root 디스크로 부팅이 포함되도록 합니다. 네트워크에서 먼저 부팅한 다음, root 디스크에서 부팅하도록 부팅 순서를 구성합니다.

director가 root 디스크로 사용할 특정 디스크를 식별합니다. **openstack overcloud deploy** 명령을 실행하면 director가 오버클라우드 이미지를 프로비저닝하고 root 디스크에 씁니다.

6.7. OVERCLOUD-MINIMAL 이미지를 사용하여 RED HAT 서브스크립션 인타이틀먼트 사용 방지

기본적으로 director는 프로비저닝 프로세스 중에 QCOW2 **overcloud-full** 이미지를 root 디스크에 씁니다. **overcloud-full** 이미지는 유효한 Red Hat 서브스크립션을 사용합니다. 그러나 예를 들어 **overcloud-minimal** 이미지를 사용하여 다른 OpenStack 서비스를 실행하지 않으려는 베어 OS를 프로비저닝하고 서브스크립션 인타이틀먼트를 사용할 수 있습니다.

이에 대한 가장 일반적인 사용 사례는 Ceph 데몬으로만 노드를 프로비저닝하는 경우 발생합니다. 이 경우와 유사한 사용 사례의 경우 **overcloud-minimal** 이미지 옵션을 사용하여 Red Hat 서브스크립션 구매 한도에 도달하지 않도록 할 수 있습니다. **overcloud-minimal** 이미지를 가져오는 방법에 관한 자세한 내용은 Obtaining images for overcloud nodes 를 참조하십시오.

절차

1. **overcloud-minimal** 이미지를 사용하도록 director를 구성하려면 다음 이미지 정의가 포함된 환경 파일을 생성합니다.

```
parameter_defaults:
  <roleName>Image: overcloud-minimal
```

2. **<roleName>**을 역할 이름으로 바꾸고 **Image**를 역할의 이름에 추가합니다. 다음 예에서는 Ceph 스토리지 노드의 **overcloud-minimal** 이미지를 보여줍니다.

```
parameter_defaults:
  CephStorageImage: overcloud-minimal
```

3. **openstack overcloud deploy** 명령에 환경 파일을 전달합니다.

참고

overcloud-minimal 이미지는 OVS가 아닌 표준 Linux 브릿지만 지원합니다. OVS는 OpenStack 서브스크립션 인타이틀먼트가 필요한 OpenStack 서비스이기 때문입니다.

6.8. 노드 수 및 플레이버를 정의하는 환경 파일 생성

기본적으로 director는 **baremetal**플레이버를 사용하여 컨트롤러 노드와 컴퓨팅 노드가 각각 1개인 오버클라우드를 배포합니다. 그러나 이 오버클라우드는 개념 검증 배포에만 적합합니다. 노드와 플레이버 수를 다르게 지정하여 기본 구성을 재정의할 수 있습니다. 소규모 프로덕션 환경의 경우 컨트롤러 노드와 컴퓨

팅 노드가 3개 이상 있고 특정 플레이버를 할당하여 노드를 적절한 리소스 사양으로 생성하는 것이 좋습니다. 다음 절차에서는 노드 수와 플레이버 할당을 저장하는 **node-info.yaml** 이라는 환경 파일을 생성하는 방법을 설명합니다.

1. **/home/stack/templates/** 디렉터리에 **node-info.yaml** 파일을 생성합니다.

```
(undercloud) $ touch /home/stack/templates/node-info.yaml
```

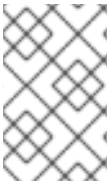
2. 필요한 노드 수 및 플레이버를 포함하도록 파일을 편집합니다. 이 예에서는 컨트롤러 노드 3개, 컴퓨팅 노드 3개, Ceph Storage 노드 3개를 배포합니다.

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudCephStorageFlavor: ceph-storage
  ControllerCount: 3
  ComputeCount: 3
  CephStorageCount: 3
```

이 파일은 6.12절. "오버클라우드 생성에 환경 파일 포함" 에서 나중에 사용됩니다.

6.9. 언더클라우드 CA를 신뢰하도록 오버클라우드 노드 구성

언더클라우드에서 TLS를 사용하고 CA가 공개적으로 신뢰되지 않은 경우 다음 절차를 따라야 합니다. 언더클라우드는 SSL 끝점 암호화에 대해 자체 CA(인증 기관)를 작동합니다. 배포의 나머지 부분에서 언더클라우드 엔드포인트에 액세스할 수 있도록 하려면 언더클라우드 CA를 신뢰하도록 오버클라우드 노드를 구성합니다.



참고

이 방법이 작동하려면 오버클라우드 노드에 언더클라우드의 공용 엔드포인트에 대한 네트워크 경로가 필요합니다. 스파인-리프형 네트워킹을 사용하는 배포에는 이 구성을 적용해야 합니다.

언더클라우드 인증서 이해

언더클라우드에서 사용할 수 있는 사용자 정의 인증서에는 사용자 제공 인증서 및 자동 생성된 인증서의 두 가지 유형이 있습니다.

- 사용자 제공 인증서 - 이 정의는 사용자가 고유 인증서를 제공한 경우에 적용됩니다. 고유 CA의 인증서이거나 자체 서명된 인증서일 수 있습니다. **undercloud_service_certificate** 옵션을 사용하여 전달됩니다. 이 경우 배포에 따라 자체 서명된 인증서 또는 CA를 신뢰해야 합니다.

- 자동 생성 인증서 - 이 정의는 **certmonger** 를 사용하여 자체 로컬 CA를 사용하여 인증서를 생성할 때 적용됩니다. 이는 **generate_service_certificate** 옵션을 사용하여 활성화됩니다. 이 경우 CA 인증서(**/etc/pki/ca-trust/source/anchors/cm-local-ca.pem**)가 있으며 언더클라우드의 HAProxy 인스턴스에서 사용하는 서버 인증서가 있습니다. 이 인증서를 OpenStack에 제공하려면 **inject-trust-anchor-hiera.yaml** 파일에 CA 인증서를 추가해야 합니다.

undercloud_service_certificate 및 **generate_service_certificate** 옵션의 설명 및 사용은 4.9절. "director 설정 매개변수" 을 참조하십시오.

언더클라우드에서 사용자 정의 인증서 사용

이 예에서는 **/home/stack/ca.crt.pem**에 있는 자체 서명 인증서를 사용합니다. 자동 생성 인증서를 사용하는 경우 대신 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** 을 사용해야 합니다.

1. 인증서 파일을 열고 인증서 부분만 복사합니다. 키를 포함하지 마십시오.

```
$ vi /home/stack/ca.crt.pem
```

필요한 인증서 부분은 다음의 단축된 예와 비슷합니다.

```
-----BEGIN CERTIFICATE-----
MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
wH
UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
-----END CERTIFICATE-----
```

2. 다음 내용으로 **/home/stack/inject-trust-anchor-hiera.yaml**이라는 새 YAML 파일을 만들고, PEM 파일에서 복사한 인증서를 추가합니다.

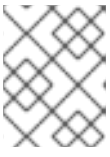
```
parameter_defaults:
  CAMap:
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV

BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
wH

UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
        -----END CERTIFICATE-----
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV

BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
wH

UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
        -----END CERTIFICATE-----
```



참고

인증서 문자열이 PEM 포멧을 따라야 하며 **content** 매개변수 내에서 올바른 YAML 들여쓰기를 사용해야 합니다.

오버클라우드 배포 중에 CA 인증서가 각 오버클라우드 노드에 복사되어 언더클라우드의 SSL 끝점에서 제공하는 암호화를 신뢰하게 됩니다. 환경 파일을 포함하는 방법에 대한 자세한 내용은 6.12절. "오버클라우드 생성에 환경 파일 포함" 을 참조하십시오.

6.10. 환경 파일을 사용하여 오버클라우드 사용자 정의

언더클라우드에는 오버클라우드 생성 계획 역할을 하는 Heat 템플릿 세트가 포함되어 있습니다. 코어

Heat 템플릿 컬렉션의 매개변수와 리소스를 재정의하는 YAML 포맷 파일인 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다. 필요한 개수만큼 많은 환경 파일을 추가할 수 있지만, 차후에 실행되는 환경 파일에 정의된 매개변수와 리소스가 우선순위를 갖기 때문에 환경 파일의 순서가 중요합니다. 다음 목록은 환경 파일 순서의 예입니다.

- 각 역할 및 해당 플레이버당 노드의 크기입니다. 오버클라우드 생성에 이 정보를 포함하는 것이 중요합니다.
- 컨테이너화된 OpenStack 서비스의 컨테이너 이미지 위치. 이는 *5장. 컨테이너 이미지 소스 구성* 의 옵션 중 하나에서 생성된 파일입니다.
- heat 템플릿 컬렉션의 초기화 파일(**environments/network-isolation.yaml**)부터 시작하여 네트워크 분리 파일, 사용자 지정 NIC 구성 파일, 마지막으로 추가 네트워크 설정 순입니다.
- 외부 로드 밸런서를 사용하는 경우 모든 외부 로드 밸런싱 환경 파일 자세한 내용은 "External Load Balancing for the Overcloud" 를 참조하십시오.
- Ceph Storage, NFS, iSCSI 등과 같은 스토리지 환경 파일
- Red Hat CDN 또는 Satellite 등록의 환경 파일 자세한 내용은 "Overcloud Registration" 을 참조하십시오.
- 기타 사용자 지정 환경 파일

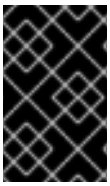
참고

/usr/share/openstack-tripleo-heat-templates/environments 디렉터리에 컨테이너화된 서비스(**docker.yaml** 및 **docker-ha.yaml**)를 활성화하는 환경 파일이 포함되어 있습니다. OpenStack Platform director에는 오버클라우드 배포 중에 이러한 파일이 자동으로 포함됩니다. 배포 명령을 사용하여 이러한 파일을 수동으로 포함하지 마십시오.

사용자 지정 환경 파일을 **templates** 디렉터리와 같은 별도의 디렉터리에 구성하는 것이 좋습니다.

Advanced Overcloud Customization 가이드를 사용하여 오버클라우드의 고급 기능을 사용자 지정할 수 있습니다.

Heat 템플릿 및 환경 파일에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Heat 템플릿 이해 섹션을 참조하십시오.



중요

기본 오버클라우드는 블록 스토리지에 지원되지 않는 구성인 로컬 LVM 스토리지를 사용합니다. 블록 스토리지에 Red Hat Ceph Storage와 같은 외부 스토리지 솔루션을 사용하는 것이 좋습니다.

6.11. CLI 툴을 사용하여 오버클라우드 생성

OpenStack 환경 생성의 최종 단계는 **openstack overcloud deploy** 명령을 실행하여 생성하는 것입니다. 이 명령을 실행하기 전에 주요 옵션 및 사용자 지정 환경 파일을 포함하는 방법을 숙지하고 있어야 합니다.

주의

openstack overcloud deploy를 백그라운드 프로세스로 실행하지 마십시오. 백그라운드 프로세스로 시작된 경우 오버클라우드 생성이 배포 도중 중단될 수 있습니다.

오버클라우드 매개변수 설정

다음 표에는 **openstack overcloud deploy** 명령을 사용할 때 추가 매개변수가 나열되어 있습니다.

표 6.2. 배포 매개 변수

매개변수	설명
--templates [TEMPLATES]	배포할 Heat 템플릿이 포함된 디렉터리. 비어 있을 경우 이 명령은 **/usr/share/openstack-tripleo-heat-templates/**를 기본 템플릿 위치로 사용합니다.
--stack STACK	생성하거나 업데이트할 스택의 이름
-t [TIMEOUT], --timeout [TIMEOUT]	배포 제한 시간(분)입니다. 이 옵션을 keystone 토큰 시간 제한보다 높은 값으로 설정하지 마십시오. 기본적으로240분입니다.
--libvirt-type [LIBVIRT_TYPE]	하이퍼바이저에 사용할 가상화 유형
--ntp-server [NTP_SERVER]	시간 동기화에 사용할 NTP(Network Time Protocol) 서버. 콤마로 구분된 목록에 여러 개의 NTP 서버를 지정할 수도 있습니다(예: **--ntp-server 0.centos.pool.org,1.centos.pool.org**). 고가용성 클러스터 배포를 위해서는 컨트롤러가 동일한 시간 소스를 일관되게 참조해야 합니다. 일반적인 환경에는 기존의 사례대로 이미 지정된 NTP 시간 소스가 있을 수 있습니다.
--no-proxy [NO_PROXY]	환경 변수 no_proxy의 사용자 정의 값을 정의합니다. 이 변수는 프록시 통신에서 특정 호스트 이름을 제외합니다.
--overcloud-ssh-user OVERCLOUD_SSH_USER	오버클라우드 노드에 액세스할 SSH 사용자를 정의합니다. 일반적으로 SSH 액세스는 **heat-admin** 사용자를 통해 실행됩니다.
-e [EXTRA HEAT TEMPLATE], --extra-template [EXTRA HEAT TEMPLATE]	오버클라우드 배포에 전달할 추가 환경 파일입니다. 두 번 이상 지정할 수 있습니다. **openstack overcloud deploy** 명령에 전달되는 환경 파일의 순서가 중요합니다. 예를 들어 순차적으로 전달되는 각 환경 파일의 매개변수는 이전 환경 파일의 동일한 매개변수를 재정의합니다.

매개변수	설명
--environment-directory	배포에 포함할 환경 파일이 들어 있는 디렉터리입니다. 이 명령은 이러한 환경 파일을 먼저 숫자 순으로 처리한 다음 알파벳순으로 처리합니다.
--validation-errors-nonfatal	오버클라우드 생성 프로세스에서는 일련의 사전 배포 검사를 수행합니다. 이 옵션은 사전 배포 검사에서 치명적이지 않은 오류가 발생할 경우에만 종료됩니다. 오류가 있으면 배포에 실패할 수 있으므로 이 옵션을 사용하는 것이 좋습니다.
--validation-warnings-fatal	오버클라우드 생성 프로세스에서는 일련의 사전 배포 검사를 수행합니다. 이 옵션은 사전 배포 검사에서 심각하지 않은 경고가 발생할 경우에만 종료됩니다.
--dry-run	오버클라우드에서 검증 확인을 수행하지만, 오버클라우드를 실제로 생성하지는 않습니다.
--skip-postconfig	오버클라우드 배포 후 설정을 건너뜁니다.
--force-postconfig	오버클라우드 배포 후 설정을 강제 적용합니다.
--skip-deploy-identifier	**DeployIdentifier** 매개변수에 대한 고유 식별자 생성을 건너뜁니다. 소프트웨어 구성 배포 단계는 구성이 실제로 변경되는 경우에만 트리거됩니다. 특정 역할 확장과 같은 소프트웨어 구성을 실행할 필요가 없다는 확신이 있을 때만 이 옵션을 신중하게 사용합니다.
--answers-file ANSWERS_FILE	인수 및 매개변수를 사용한 YAML 파일의 경로입니다.
--rhel-reg	오버클라우드 노드를 고객 포털 또는 Satellite 6에 등록합니다.
--reg-method	오버클라우드 노드에 사용할 등록 방법입니다. Red Hat Satellite 6 또는 Red Hat Satellite 5에는 **satellite**를 고객 포털에는 **portal**을 사용합니다.
--reg-org [REG_ORG]	등록에 사용하는 조직입니다.
--reg-force	시스템이 이미 등록되어 있어도 등록합니다.

매개변수	설명
--reg-sat-url [REG_SAT_URL]	오버클라우드 노드를 등록할 Satellite 서버의 기본 URL입니다. Satellite의 HTTPS URL 대신 HTTP URL을 이 매개변수에 사용합니다. 예를 들어 https://satellite.example.com 대신 http://satellite.example.com을 사용합니다. 오버클라우드 생성 프로세스에서는 이 URL을 사용하여 서버가 Red Hat Satellite 5 서버인지 아니면 Red Hat Satellite 6 서버인지 확인합니다. Red Hat Satellite 6 서버인 경우 오버클라우드는 **katello-ca-consumer-latest.noarch.rpm** 파일을 가져오고 **subscription-manager** 에 등록한 후 **katello-agent** 를 설치합니다. Red Hat Satellite 5 서버의 경우 오버클라우드는 **RHN-ORG-TRUSTED-SSL-CERT** 파일을 가져오고 **rhnreg_ks** 에 등록합니다.
--reg-activation-key [REG_ACTIVATION_KEY]	등록에 사용할 활성화 키입니다.

일부 명령행 매개변수는 오래되었거나 더 이상 사용되지 않으며, 대신 환경 파일의 **parameter_defaults** 섹션에 포함하는 Heat 템플릿 매개변수가 사용됩니다. 다음 표에는 더 이상 사용되지 않는 매개변수가 해당하는 Heat 템플릿 매개변수에 매핑되어 있습니다.

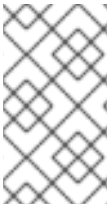
표 6.3. 더 이상 사용되지 않는 CLI 매개변수와 해당하는 Heat 템플릿 매개변수 매핑

매개변수	설명	Heat 템플릿 매개변수
--control-scale	확장할 컨트롤러 노드 수	**ControllerCount**
--compute-scale	확장할 컴퓨팅 노드 수	**ComputeCount**
--ceph-storage-scale	확장할 Ceph Storage 노드 수	**CephStorageCount**
--block-storage-scale	확장할 Cinder 노드 수	**BlockStorageCount**
--swift-storage-scale	확장할 Swift 노드 수	**ObjectStorageCount**
--control-flavor	컨트롤러 노드에 사용할 플레이버	**OvercloudControllerFlavor**
--compute-flavor	컴퓨팅 노드에 사용할 플레이버	**OvercloudComputeFlavor**
--ceph-storage-flavor	Ceph Storage 노드에 사용할 플레이버	**OvercloudCephStorageFlavor**
--block-storage-flavor	Cinder 노드에 사용할 플레이버	**OvercloudBlockStorageFlavor**

매개변수	설명	Heat 템플릿 매개변수
--swift-storage-flavor	Swift Storage 노드에 사용할 플레이버	**OvercloudSwiftStorageFlavor**
--neutron-flat-networks	neutron 플러그인에 구성할 플랫 네트워크를 정의합니다. 외부 네트워크 생성을 허용하도록 기본적으로 "datacentre"로 설정	**NeutronFlatNetworks**
--neutron-physical-bridge	각 하이퍼바이저에 생성할 Open vSwitch 브리지입니다. 기본값은 "br-ex"입니다. 일반적으로 이 값은 변경할 필요가 없습니다.	**HypervisorNeutronPhysicalBridge**
--neutron-bridge-mappings	사용할 논리적 브릿지와 물리적 브릿지의 매핑. 기본적으로 호스트(br-ex)의 외부 브릿지를 물리 이름(datacentre)에 매핑합니다. 기본 유동 네트워크에 이 값을 사용합니다.	**NeutronBridgeMappings**
--neutron-public-interface	네트워크 노드의 br-ex에 브릿지에 인터페이스를 정의합니다.	**NeutronPublicInterface**
--neutron-network-type	Neutron의 테넌트 네트워크 유형	**NeutronNetworkType**
--neutron-tunnel-types	Neutron 테넌트 네트워크의 터널 유형. 여러 값을 지정하려면 쉼표로 구분된 문자열을 사용합니다.	**NeutronTunnelTypes**
--neutron-tunnel-id-ranges	테넌트 네트워크 할당에 사용할 수 있는 GRE 터널 ID 범위	**NeutronTunnelIdRanges**
--neutron-vni-ranges	테넌트 네트워크 할당에 사용할 수 있는 VXLAN VNI ID 범위	**NeutronVniRanges**
--neutron-network-vlan-ranges	지원할 Neutron ML2 및 Open vSwitch VLAN 매핑 범위입니다. 기본적으로 *datacentre* 물리적 네트워크에서 VLAN을 허용하도록 설정되어 있습니다.	**NeutronNetworkVLANRanges**
--neutron-mechanism-drivers	neutron 테넌트 네트워크의 메커니즘 드라이버입니다. 기본값은 "openvswitch"입니다. 여러 값을 지정하려면 쉼표로 구분된 문자열을 사용합니다.	**NeutronMechanismDrivers**

매개변수	설명	Heat 템플릿 매개변수
--neutron-disable-tunneling	VLAN 세그먼트화된 네트워크 또는 플랫 네트워크를 Neutron과 함께 사용하려는 경우 터널링을 비활성화합니다.	매개 변수 매핑 없음
--validation-errors-fatal	오버클라우드 생성 프로세스에서는 일련의 사전 배포 검사를 수행합니다. 이 옵션은 사전 배포 확인에서 치명적인 오류가 발생할 경우에만 종료됩니다. 오류가 있으면 배포에 실패할 수 있으므로 이 옵션을 사용하는 것이 좋습니다.	매개변수 매핑 없음

이러한 매개변수는 Red Hat OpenStack Platform의 이후 버전에서 삭제될 예정입니다.



참고

전체 옵션 목록을 보려면 다음 명령을 실행합니다.

```
(undercloud) $ openstack help overcloud deploy
```

6.12. 오버클라우드 생성에 환경 파일 포함

-e 에는 오버클라우드를 사용자 정의할 환경 파일이 포함되어 있습니다. 환경 파일은 필요한 수만큼 추가할 수 있습니다. 차후에 실행되는 환경 파일에 정의된 매개변수와 리소스가 우선순위를 갖기 때문에 환경 파일의 순서가 중요합니다. 다음 목록은 환경 파일 순서의 예입니다.

- 각 역할 및 해당 플레이버당 노드의 크기입니다. 오버클라우드 생성에 이 정보를 포함하는 것이 중요합니다.
- 컨테이너화된 OpenStack 서비스의 컨테이너 이미지 위치. 이는 5장. *컨테이너 이미지 소스 구성* 의 옵션 중 하나에서 생성된 파일입니다.
- heat 템플릿 컬렉션의 초기화 파일(**environments/network-isolation.yaml**)부터 시작하여 네트워크 분리 파일, 사용자 지정 NIC 구성 파일, 마지막으로 추가 네트워크 설정 순입니다.
- 외부 로드 밸런서를 사용하는 경우 모든 외부 로드 밸런싱 환경 파일 자세한 내용은 "External Load Balancing for the Overcloud" 를 참조하십시오.
- Ceph Storage, NFS, iSCSI 등과 같은 스토리지 환경 파일
- Red Hat CDN 또는 Satellite 등록의 환경 파일 자세한 내용은 "Overcloud Registration" 을 참조하십시오.
- 기타 사용자 지정 환경 파일

참고

/usr/share/openstack-tripleo-heat-templates/environments 디렉터리에 컨테이너화된 서비스(**docker.yaml** 및 **docker-ha.yaml**)를 활성화하는 환경 파일이 포함되어 있습니다. OpenStack Platform director에는 오버클라우드 배포 중에 이러한 파일이 자동으로 포함됩니다. 배포 명령을 사용하여 이러한 파일을 수동으로 포함하지 마십시오.

-e 옵션을 사용하여 오버클라우드에 추가된 환경 파일은 오버클라우드 스택 정의의 일부가 됩니다. 다음 명령은 사용자 지정 환경 파일을 사용하여 오버클라우드 생성을 시작하는 방법의 예입니다.

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/node-info.yaml\
  -e /home/stack/templates/overcloud_images.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml \
  -e /home/stack/templates/ceph-custom-config.yaml \
  -e /home/stack/inject-trust-anchor-hiera.yaml \
  -r /home/stack/templates/roles_data.yaml \
  --ntp-server pool.ntp.org \
```

이 명령에는 다음과 같은 추가 옵션을 사용할 수 있습니다.

--templates

/usr/share/openstack-tripleo-heat-templates의 Heat 템플릿 컬렉션을 기반으로 하여 오버클라우드를 생성합니다.

-e /home/stack/templates/node-info.yaml

각 역할에 사용할 노드 수와 유형을 정의하는 환경 파일을 추가합니다. 예를 들면 다음과 같습니다.

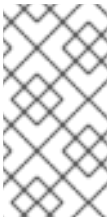
```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudCephStorageFlavor: ceph-storage
  ControllerCount: 3
  ComputeCount: 3
  CephStorageCount: 3
```

-e /home/stack/templates/overcloud_images.yaml

컨테이너 이미지 소스가 포함된 환경 파일을 추가합니다. 자세한 내용은 5장. *컨테이너 이미지 소스 구성*를 참조하십시오.

-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml

환경 파일을 추가하여 오버클라우드 배포에서 네트워크 분리를 초기화합니다.

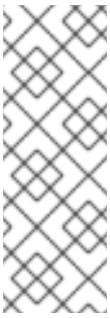


참고

network-isolation.j2.yaml 은 이 템플릿의 Jinja2 버전입니다. **openstack overcloud deploy** 명령은 Jinja2 템플릿을 일반 YAML 파일로 렌더링합니다. 따라서 **openstack overcloud deploy** 명령을 실행할 때 결과 렌더링된 YAML 파일 이름(이 경우 **network-isolation.yaml**)을 포함해야 합니다.

-e /home/stack/templates/network-environment.yaml

네트워크 분리를 사용자 지정하는 환경 파일을 추가합니다.



참고

openstack overcloud netenv validate 명령을 실행하여 **network-environment.yaml** 파일의 구문을 검증합니다. 또한 이 명령은 컴퓨팅, 컨트롤러, 스토리지 및 구성 가능한 역할 네트워크 파일에 대한 개별 nic-config 파일의 유효성을 검사합니다. **-f** 또는 **--file** 옵션을 사용하여 검증할 파일을 지정합니다.

```
$ openstack overcloud netenv validate -f ~/templates/network-environment.yaml
```

-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml

Ceph Storage 서비스를 활성화하는 환경 파일을 추가합니다.

-e /home/stack/templates/ceph-custom-config.yaml

Ceph Storage 구성을 사용자 정의하는 환경 파일을 추가합니다.

-e /home/stack/inject-trust-anchor-hiera.yaml

환경 파일을 추가하여 언더클라우드에 사용자 지정 인증서를 설치합니다.

--ntp-server pool.ntp.org

시간 동기화에 NTP 서버를 사용합니다. 이는 컨트롤러 노드 클러스터를 동기화 상태로 유지하는 데 필요합니다.

-r /home/stack/templates/roles_data.yaml

(선택 사항) 사용자 지정 역할 또는 다중 아키텍처 클라우드를 사용하는 경우 생성된 역할 데이터입니다. 자세한 내용은 6.4절. "아키텍처별 역할 생성"를 참조하십시오.

director를 사용하려면 9장. *오버클라우드 생성 후 작업 수행* 에서 재배포 및 배포 후 기능을 위해 이러한 환경 파일이 필요합니다. 이러한 파일을 포함하지 않으면 오버클라우드가 손상될 수 있습니다.

오버클라우드 설정을 나중에 수정하려는 경우 다음을 수행해야 합니다.

1. 사용자 지정 환경 파일 및 Heat 템플릿에서 매개변수 수정
2. 같은 환경 파일을 지정하고 **openstack overcloud deploy** 명령 다시 실행

환경 파일 디렉터리 포함

environment **-directory 옵션을 사용하여 환경 파일이 포함된 전체 디렉터리를** 추가할 수 있습니다. 배포 명령은 이 디렉터리의 환경 파일을 숫자 순으로 처리한 다음 알파벳순으로 처리합니다. 이 방법을 사용하는 경우 숫자 접두사가 있는 파일 이름을 사용하여 처리 방법을 주문하는 것이 좋습니다. 예를 들면 다음과 같습니다.

```
(undercloud) $ ls -1 ~/templates
00-node-info.yaml
10-overcloud_images.yaml
20-network-isolation.yaml
30-network-environment.yaml
40-storage-environment.yaml
50-rhel-registration.yaml
```

다음 배포 명령을 실행하여 디렉터리를 포함합니다.

```
(undercloud) $ openstack overcloud deploy --templates --environment-directory ~/templates
```

answers 파일 사용

응답 파일은 템플릿 및 환경 파일을 간단하게 포함할 수 있는 YAML 형식 파일입니다. 응답 파일은 다음 매개 변수를 사용합니다.

templates

사용할 코어 Heat 템플릿 컬렉션입니다. 이는 **--templates** 명령줄 옵션을 대체하는 역할을 합니다.

환경

포함할 환경 파일 목록입니다. 이는 **--environment-file** (**-e**) 명령줄 옵션을 대체하는 역할을 합니다.

예를 들어 응답 파일에는 다음이 포함될 수 있습니다.

```
templates: /usr/share/openstack-tripleo-heat-templates/
environments:
  - ~/templates/00-node-info.yaml
  - ~/templates/10-network-isolation.yaml
  - ~/templates/20-network-environment.yaml
  - ~/templates/30-storage-environment.yaml
  - ~/templates/40-rhel-registration.yaml
```

응답 파일을 포함하도록 다음 배포 명령을 실행합니다.

```
(undercloud) $ openstack overcloud deploy --answers-file ~/answers.yaml
```

오버클라우드 구성 및 환경 파일 관리 지침

다음 지침에 따라 환경 파일 및 오버클라우드 설정을 관리할 수 있습니다.

- 코어 heat 템플릿을 직접 수정하지 마십시오. 이로 인해 바람직하지 않은 결과가 발생하고 환경을 손상시킬 수 있습니다. 환경 파일을 통해 오버클라우드 설정을 수정합니다.
- 오버클라우드 설정을 직접 편집하지 마십시오. 오버클라우드 스택을 director로 업데이트할 때 수동 설정을 director의 설정이 덮어쓰므로 설정을 직접 편집하지 마십시오. 환경 파일을 통해 오버클라우드 설정을 수정하고 배포 명령을 다시 실행합니다.
- 배포 명령을 포함하는 bash 스크립트를 생성하고 오버클라우드 업데이트를 수행할 때 이 스크립트를 사용합니다. 이 스크립트를 사용하면 **openstack overcloud deploy** 명령을 재실행할 때 정확한 옵션 및 환경 파일을 일관되게 유지하고 오버클라우드 중단을 방지할 수 있습니다.
- 원치 않는 변경을 방지하고 과거의 변경 사항을 추적하도록 환경 파일이 들어 있는 디렉터리의 버전을 유지 관리합니다.

6.13. 오버클라우드 계획 관리

openstack overcloud deploy 명령을 사용하는 대신 director에서 가져온 플랜도 관리할 수 있습니다.

새 계획을 생성하려면 **stack** 사용자로 다음 명령을 실행합니다.

```
(undercloud) $ openstack overcloud plan create --templates /usr/share/openstack-tripleo-heat-templates my-overcloud
```

그러면 **/usr/share/openstack-tripleo-heat-templates** 의 코어 Heat 템플릿 컬렉션에서 계획을 생성합니다. director는 입력을 기반으로 계획의 이름을 지정합니다. 예에서는 **my-overcloud** 입니다. director는 이 이름을 오브젝트 스토리지 컨테이너, 워크플로 환경, 오버클라우드 스택 이름의 레이블로 사용합니다.

다음 명령을 사용하여 환경 파일의 매개변수를 추가합니다.

```
(undercloud) $ openstack overcloud parameters set my-overcloud ~/templates/my-environment.yaml
```

다음 명령을 사용하여 계획을 배포합니다.

```
(undercloud) $ openstack overcloud plan deploy my-overcloud
```

다음 명령을 사용하여 기존 계획을 삭제합니다.

```
(undercloud) $ openstack overcloud plan delete my-overcloud
```

참고

openstack overcloud deploy 명령은 기본적으로 이러한 모든 명령을 사용하여 기존 계획을 제거하고, 환경 파일을 사용하여 새 계획을 업로드하고, 계획을 배포합니다.

6.14. 오버클라우드 템플릿 및 계획 검증

오버클라우드 생성 또는 스택 업데이트를 실행하기 전에 Heat 템플릿 및 환경 파일에 오류가 있는지 확인합니다.

렌더링된 템플릿 생성

오버클라우드의 코어 Heat 템플릿은 Jinja2 형식으로 되어 있습니다. 템플릿을 확인하려면 다음 명령을 사용하여 Jinja2 포맷 없이 버전을 렌더링합니다.

```
(undercloud) $ openstack overcloud plan create --templates /usr/share/openstack-tripleo-heat-templates overcloud-validation
(undercloud) $ mkdir ~/overcloud-validation
(undercloud) $ cd ~/overcloud-validation
(undercloud) $ openstack container save overcloud-validation
```

다음 검증 테스트에 **~/overcloud-validation** 에서 렌더링된 템플릿을 사용합니다.

템플릿 구문 검증

다음 명령을 사용하여 템플릿 구문의 유효성을 검사합니다.

```
(undercloud) $ openstack orchestration template validate --show-nested --template ~/overcloud-validation/overcloud.yaml -e ~/overcloud-validation/overcloud-resource-registry-puppet.yaml -e [ENVIRONMENT FILE] -e [ENVIRONMENT FILE]
```

참고

검증을 수행하려면 **overcloud-resource-registry-puppet.yaml** 환경 파일에 오버클라우드별 리소스를 포함해야 합니다. **-e** 옵션을 사용하여 이 명령에 환경 파일을 추가합니다. 또한 **--show-nested** 옵션을 포함하여 중첩된 템플릿에서 매개변수를 확인합니다.

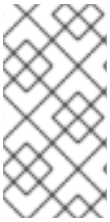
이 명령은 템플릿에서 구문 오류를 식별합니다. 템플릿 구문의 검증이 성공적으로 수행되면 출력에 결과 오버클라우드 템플릿의 프리뷰가 표시됩니다.

6.15. 오버클라우드 생성 모니터링

오버클라우드 생성 프로세스가 시작되고 director에서 노드를 프로비저닝합니다. 이 프로세스를 완료하는 데 시간이 다소 걸립니다. 오버클라우드 생성 상태를 보려면 **stack** 사용자로 별도의 터미널을 열고 다음을 실행합니다.

```
(undercloud) $ source ~/stackrc
(undercloud) $ openstack stack list --nested
```

openstack stack list --nested 명령은 오버클라우드 생성의 현재 단계를 보여줍니다.



참고

초기 오버클라우드 생성에 실패하면 **openstack stack delete overcloud** 명령을 사용하여 부분적으로 배포된 오버클라우드를 삭제하고 다시 시도할 수 있습니다. 이러한 초기 오버클라우드 생성이 실패한 경우에만 이 명령을 실행합니다. 완전히 배포된 운영 오버클라우드에서 이 명령을 실행하지 마십시오. 그렇지 않으면 전체 오버클라우드를 삭제합니다.

6.16. 오버클라우드 배포 출력 보기

오버클라우드 배포에 성공하면 쉘은 오버클라우드에 액세스하는 데 사용할 수 있는 다음 정보를 반환합니다.

```
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

6.17. 오버클라우드 액세스

director는 director 호스트에서 오버클라우드와의 상호 작용을 구성하고 인증을 지원하는 스크립트를 생성합니다. director는 **overcloudrc** 파일을 **stack** 사용자의 홈 director에 저장합니다. 이 파일을 사용하려면 다음 명령을 실행합니다.

```
(undercloud) $ source ~/overcloudrc
```

이렇게 하면 director 호스트의 CLI에서 오버클라우드와 상호 작용하는 데 필요한 환경 변수가 로드됩니다. 명령 프롬프트가 다음과 같이 변경됩니다.

```
(overcloud) $
```

director 호스트와의 상호 작용으로 돌아가려면 다음 명령을 실행합니다.

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

오버클라우드의 각 노드에는 **heat-admin** 이라는 사용자도 포함되어 있습니다. **stack** 사용자는 각 노드에서 이 사용자에 대한 SSH 액세스 권한을 갖습니다. SSH를 통해 노드에 액세스하려면 원하는 노드의 IP 주소를 확인합니다.

```
(undercloud) $ openstack server list
```

다음으로 **heat-admin** 사용자와 노드의 IP 주소를 사용하여 노드에 연결합니다.

```
(undercloud) $ ssh heat-admin@192.168.24.23
```

6.18. 오버클라우드 생성 완료

이제 명령행 툴을 사용한 오버클라우드 생성이 완료되었습니다. 생성 후 기능에 대해서는 9장. *오버클라우드 생성 후 작업 수행*에서 참조하십시오.

7장. 웹 UI로 기본 오버클라우드 구성

이 장에서는 웹 UI를 사용하는 OpenStack Platform 환경의 기본 구성 단계를 설명합니다. 기본 설정을 사용하는 오버클라우드에는 사용자 지정 기능이 없습니다. 하지만 Advanced Overcloud Customization 가이드의 지침에 따라 이 기본 오버클라우드에 고급 옵션을 추가하고 사양에 맞게 사용자 지정할 수 있습니다.

이 장의 예에서는 모든 노드가 전원 관리에 IPMI를 사용하는 베어 메탈 시스템입니다. 지원되는 추가 전원 관리 유형 및 옵션은 *부록 B. 전원 관리 드라이버* 을 참조하십시오.

워크플로

1. 노드 정의 템플릿 및 수동 등록을 사용하여 빈 노드를 등록합니다.
2. 모든 노드의 하드웨어를 검사합니다.
3. 오버클라우드 플랜을 director에 업로드합니다.
4. 노드를 역할에 할당합니다.

요구 사항

- *4장. 언더클라우드 설치* UI가 활성화된 director 노드
- 노드에 사용할 베어 메탈 시스템 세트. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다(Overcloud 역할에 대한 내용은 3.1절. "노드 배포 역할 계획" 참조). 또한 이러한 머신은 각 노드 유형에 설정된 요구 사항을 준수해야 합니다. 이러한 요구 사항은 2.4절. "오버클라우드 요구 사항" 에서 참조하십시오. 이러한 노드에는 운영 체제가 필요하지 않습니다. director는 Red Hat Enterprise Linux 7 이미지를 각 노드에 복사합니다.
- 기본 VLAN으로 구성된 프로비저닝 네트워크에 대한 하나의 네트워크 연결입니다. 모든 노드는 이 네트워크에 연결되어야 하며 2.3절. "네트워킹 요구 사항" 에 설정된 요구 사항을 준수해야 합니다.
- 다른 모든 네트워크 유형은 OpenStack 서비스에 프로비저닝 네트워크를 사용합니다. 하지만 다른 네트워크 트래픽 유형에 대해 추가 네트워크를 생성할 수 있습니다.

중요

다중 아키텍처 클라우드를 활성화하면 UI 워크플로가 지원되지 않습니다. 다음 지침을 따르십시오. *6장. CLI 툴로 기본 오버클라우드 구성*

7.1. 웹 UI 액세스

사용자는 SSL을 통해 director의 웹 UI에 액세스할 수 있습니다. 예를 들어 언더클라우드의 IP 주소가 192.168.24.1인 경우 UI에 액세스할 주소는 **https://192.168.24.1** 입니다. 웹 UI는 처음에 다음과 같은 필드가 있는 로그인 화면을 제공합니다.

- **사용자 이름** - director의 관리 사용자입니다. 기본값은 **admin** 입니다.
- **암호** - 관리 사용자의 암호입니다. 언더클라우드 호스트 터미널에서 **sudo hiera admin_password** 를 **stack** 사용자로 실행하여 암호를 확인합니다.

UI에 로그인하면 UI는 OpenStack Identity Public API에 액세스하여 다른 공용 API 서비스의 끝점을 가져옵니다. 이러한 서비스에는 다음이 포함됩니다.

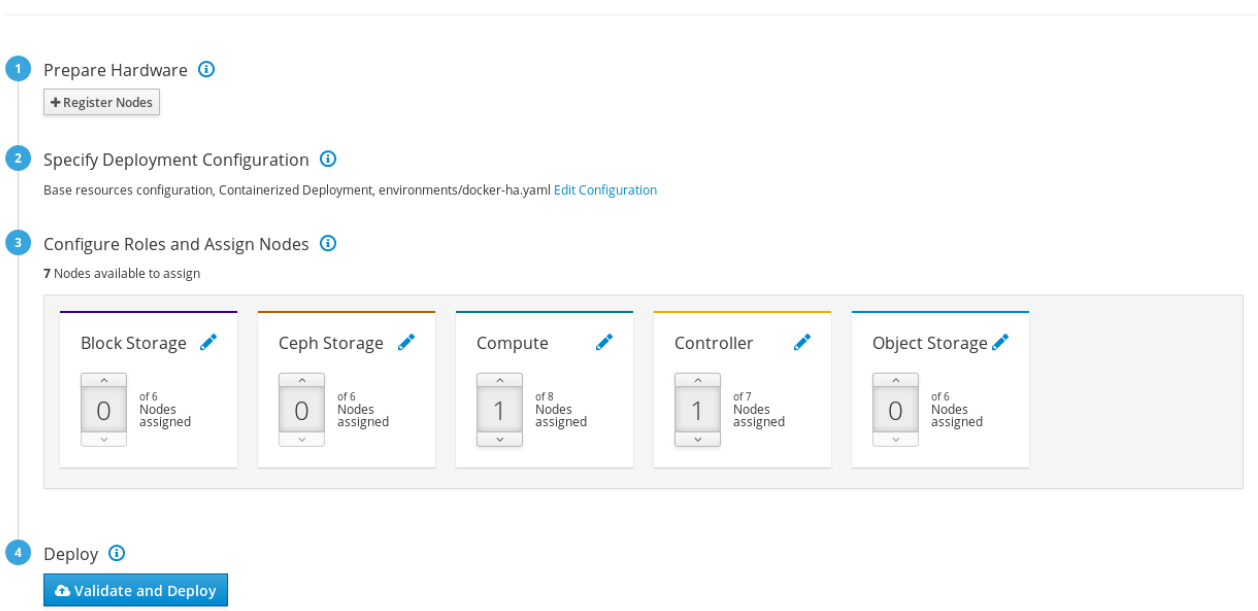
구성 요소	UI 용도
OpenStack Identity(**keystone**)	UI에 대한 인증 및 기타 서비스의 엔드포인트 검색에 사용됩니다.
OpenStack Orchestration(**heat**)	배포 상태에 해당합니다.
OpenStack Bare Metal(**ironic**)	노드 제어에 사용됩니다.
OpenStack Object Storage(빠름)	Heat 템플릿 컬렉션 또는 오버클라우드 생성에 사용되는 플랜의 저장을 위해 사용됩니다.
OpenStack Workflow(mistral)	director 작업에 액세스하고 실행합니다.
OpenStack Messaging(z**aqar**)	특정 작업의 상태를 찾는 websocket 기반 서비스입니다.

7.2. 웹 UI 탐색

UI는 세 가지 주요 섹션을 제공합니다.

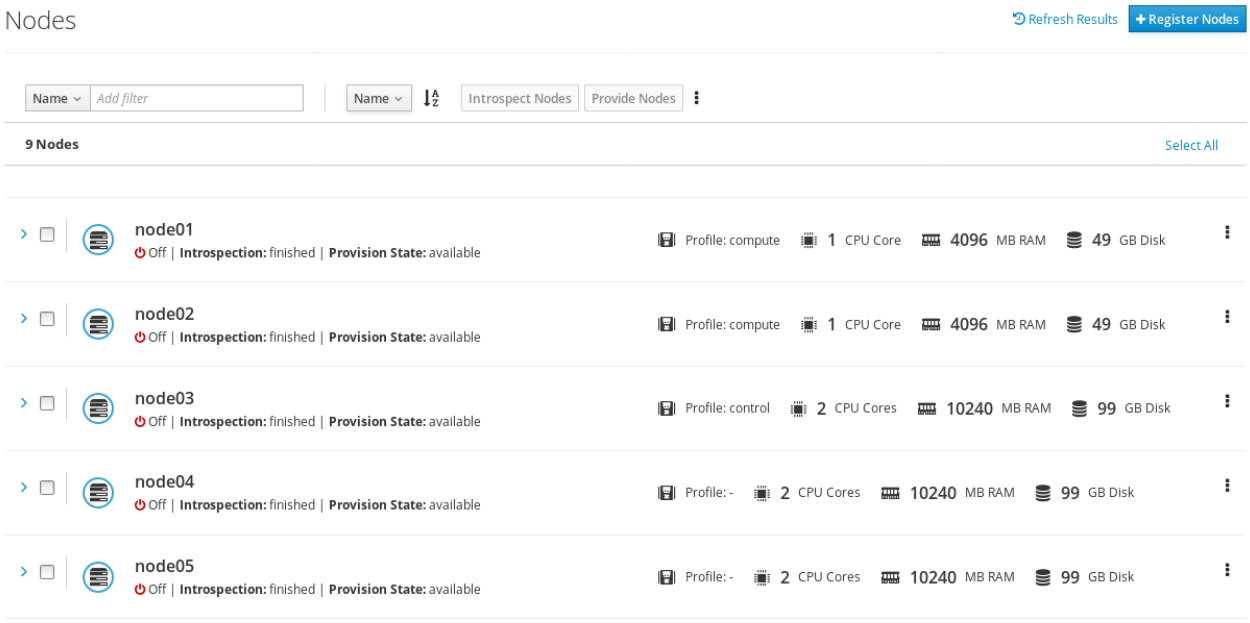
Plan

UI 상단에 있는 메뉴 항목입니다. 이 페이지는 기본 UI 섹션 역할을 하며 오버클라우드 생성에 사용할 플랜, 각 역할에 할당할 노드 및 현재 오버클라우드의 상태를 정의할 수 있습니다. 이 섹션에서는 배포 매개 변수 설정 및 역할에 노드 할당을 포함하여 오버클라우드 생성 프로세스의 각 단계를 안내하는 배포 워크플로를 제공합니다.

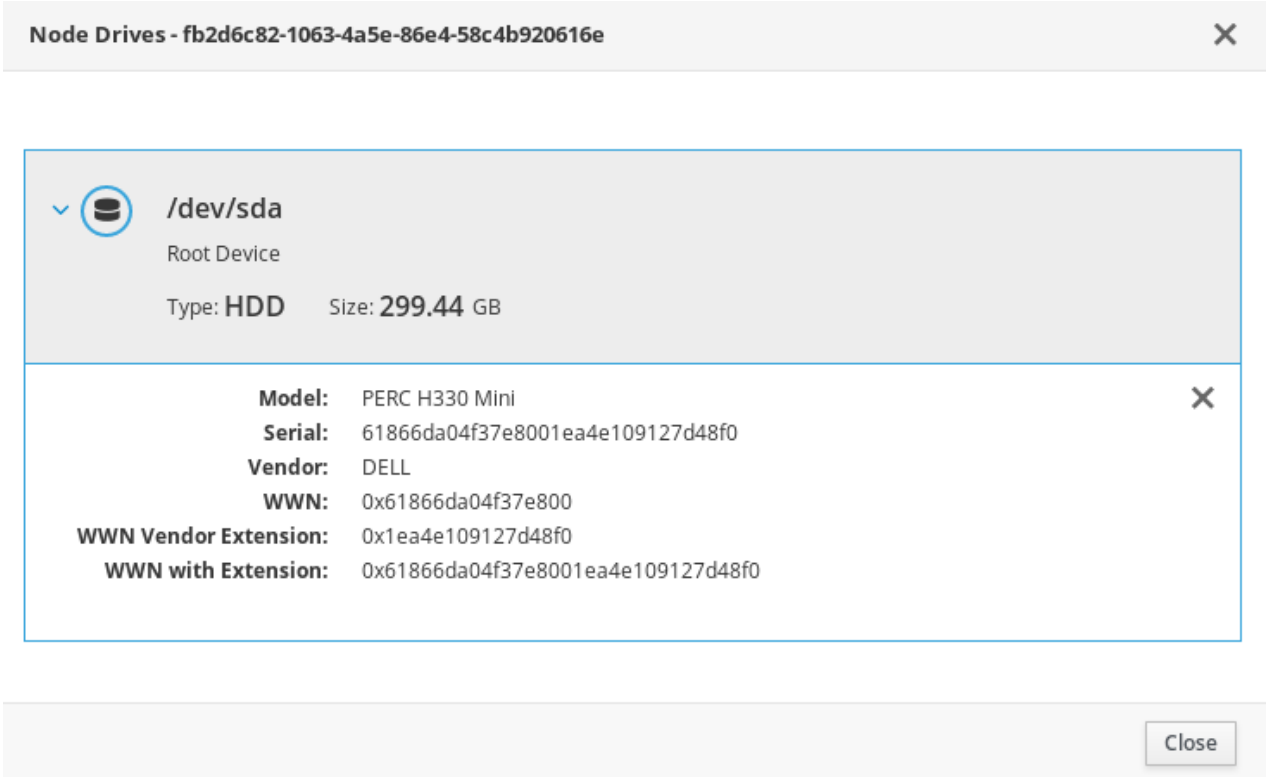


노드

UI 상단에 있는 메뉴 항목입니다. 이 페이지는 노드 구성 섹션 역할을 하며 새 노드를 등록하고 등록된 노드를 검사하는 방법을 제공합니다. 또한 이 섹션에서는 전원 상태, 인트로스펙션 상태, 프로비저닝 상태 및 하드웨어 정보와 같은 정보를 표시합니다.

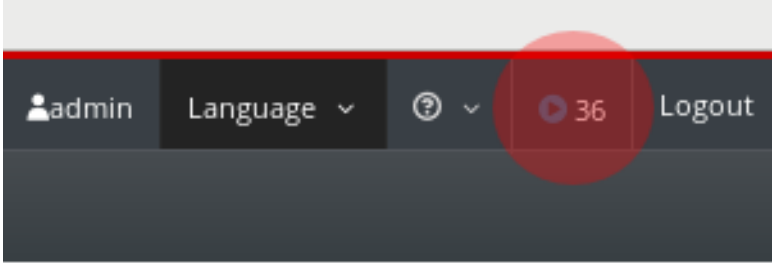


각 노드 오른쪽에 있는 오버플로 메뉴 항목(점 3개)을 클릭하면 선택한 노드에 대한 디스크 정보가 표시됩니다.



Validations

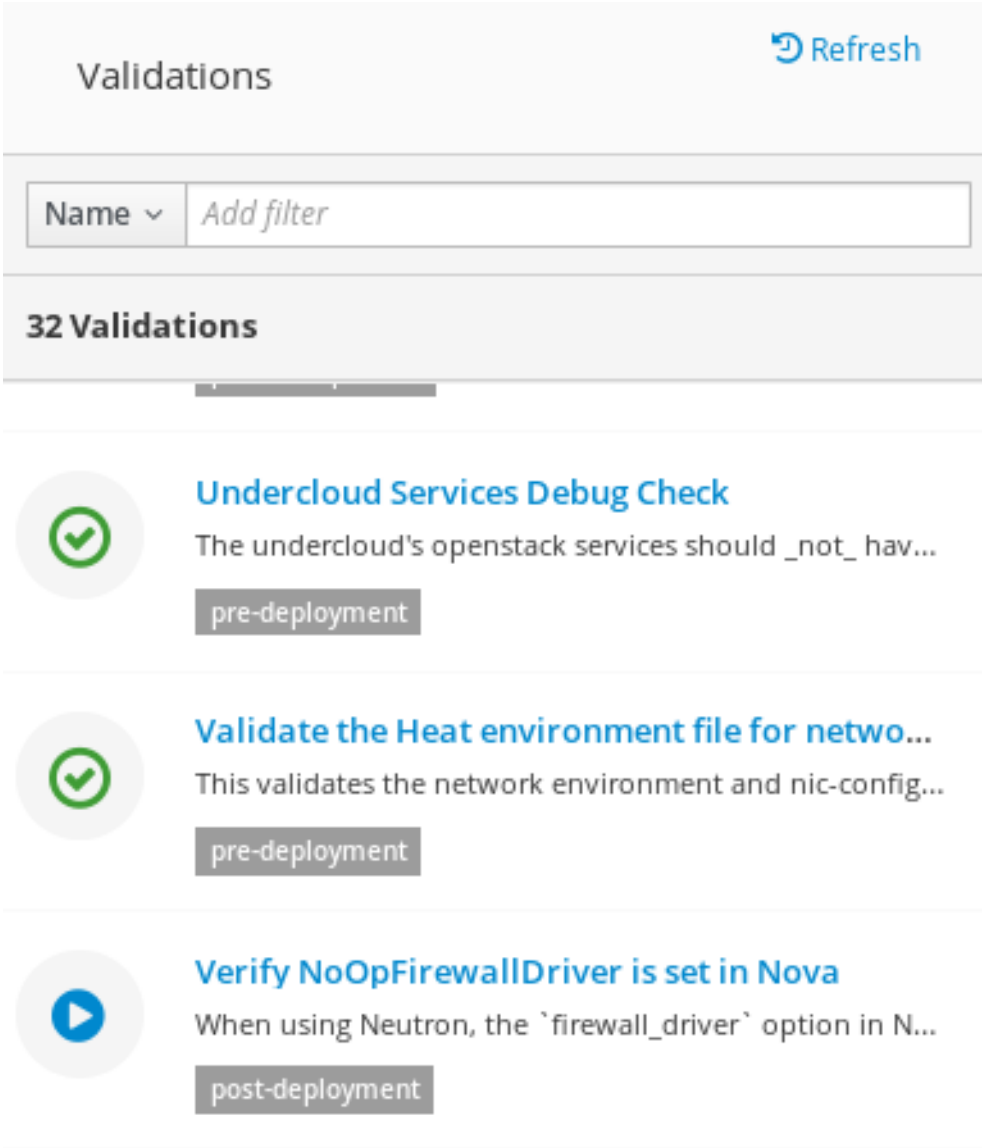
Validations 메뉴 옵션을 클릭하면 페이지 오른쪽에 패널이 표시됩니다.



이 섹션에서는 다음에 대한 일련의 시스템 검사를 제공합니다.

- 배포 전
- 배포 후
- 인트로스펙션 전
- 업그레이드 전
- 업그레이드 후

이러한 검증 작업은 배포 시 특정 시점에서 자동으로 실행됩니다. 그러나 수동으로 실행할 수도 있습니다. 실행하려는 검증 작업의 **재생** 버튼을 클릭합니다. 각 검증 작업의 제목을 클릭하여 실행하고 검증 제목을 클릭하여 자세한 정보를 확인합니다.



7.3. 웹 UI에서 오버클라우드 플랜 가져오기

director UI에는 오버클라우드를 구성하기 전에 플랜이 필요합니다. 이 계획은 일반적으로 **/usr/share/openstack-tripleo-heat-templates** 의 언더클라우드에 있는 것과 같은 Heat 템플릿 컬렉션입니다. 또한 하드웨어 및 환경 요구 사항에 맞게 플랜을 사용자 지정할 수 있습니다. 오버클라우드 사용자 지정에 대한 자세한 내용은 Advanced Overcloud Customization 가이드를 참조하십시오.

계획에는 오버클라우드 설정을 위한 네 가지 기본 단계가 표시됩니다.

1. **하드웨어 준비** - 노드 등록 및 인트로스펙션
2. **배포 구성 지정** - 오버클라우드 매개 변수 구성 및 포함할 환경 파일 정의
3. **역할 구성 및 노드 할당** - 노드를 역할에 할당하고 역할별 매개 변수를 수정합니다.
4. **배포** - 오버클라우드 생성을 시작합니다.

언더클라우드 설치 및 구성이 계획을 자동으로 업로드합니다. 웹 UI에서 여러 플랜을 가져올 수도 있습니다. 플랜 화면에서 **모든 플랜** breadcrumb를 클릭합니다. **그러면 현재 계획 목록이 표시됩니다. 카드를 클릭하여 여러 플랜 간에 변경합니다.**

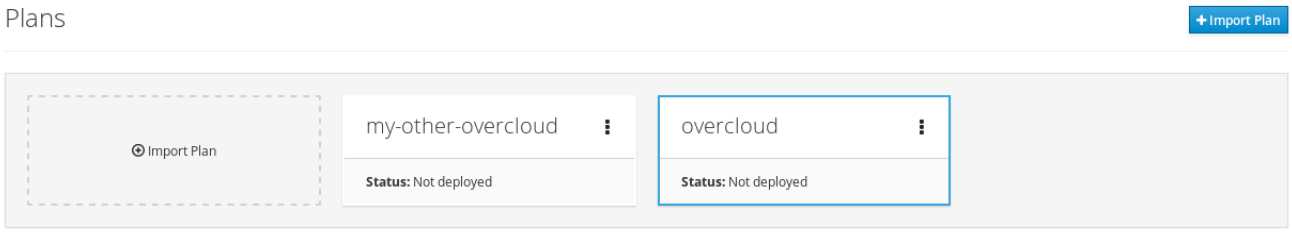
플랜 가져오기 를 클릭하면 다음 정보를 묻는 창이 표시됩니다.

- 플랜 이름 - 플랜의 일반 텍스트 이름입니다. 예:**overcloud**.
- 업로드 유형 - Tar 아카이브 (tar.gz) 또는 전체로컬 폴더 (Google Chrome 만 해당)를 업로드할지 여부를 선택합니다.
- 플랜 파일 - 브라우저를 클릭하여 로컬 파일 시스템에서 플랜을 선택합니다.

director의 Heat 템플릿 컬렉션을 클라이언트 머신에 복사해야 하는 경우 파일을 압축하여 복사합니다.

```
$ cd /usr/share/openstack-tripleo-heat-templates/
$ tar -cf ~/overcloud.tar *
$ scp ~/overcloud.tar user@10.0.0.55:~/.
```

director UI에서 플랜을 업로드하면 플랜이 플랜 목록에 표시되어 구성할 수 있습니다. 선택한 플랜 카드를 클릭합니다.

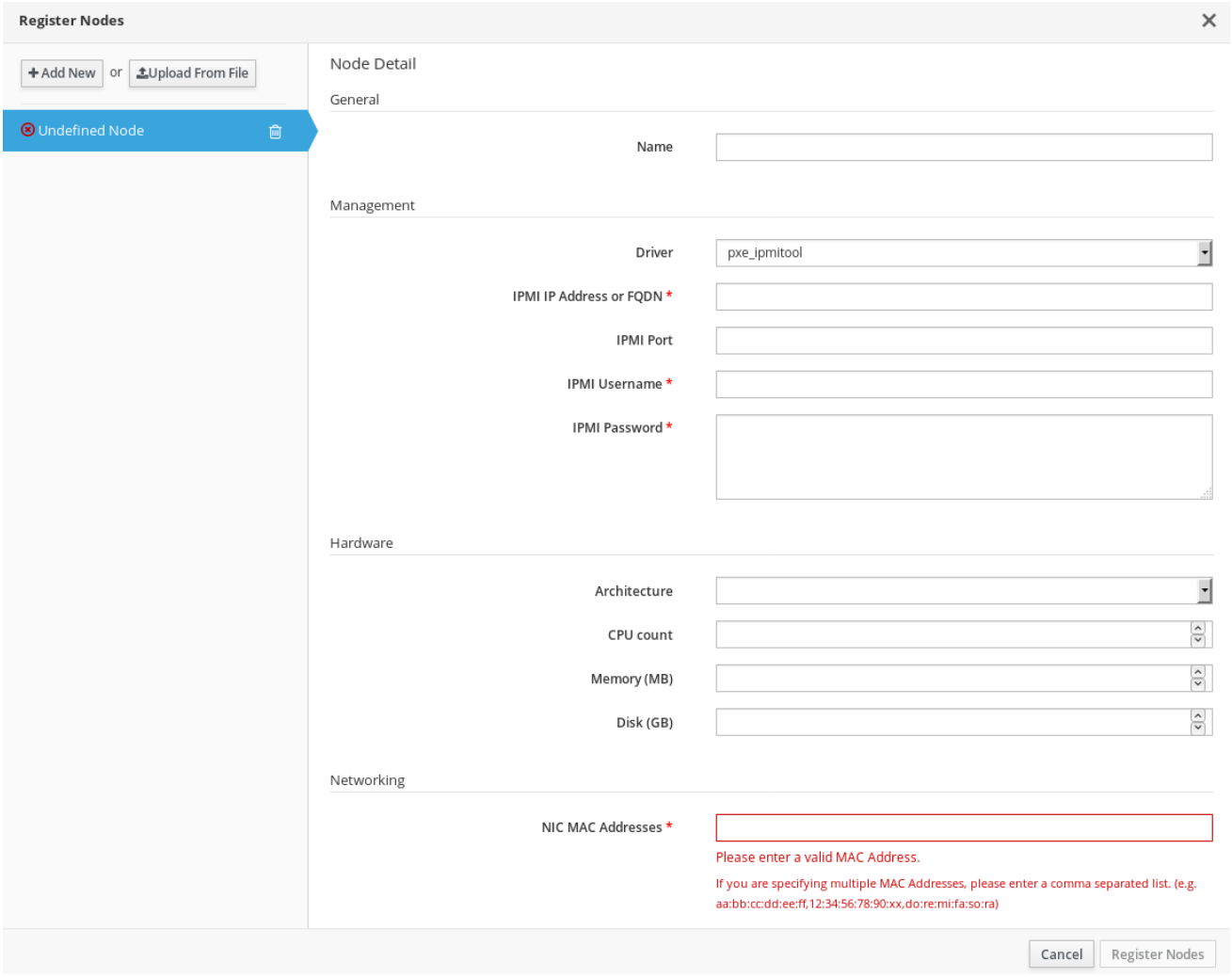


7.4. 웹 UI에서 노드 등록

오버클라우드 구성의 첫 단계는 노드를 등록하는 것입니다. 다음을 통해 노드 등록 프로세스를 시작합니다.

- 플랜 화면에서 1 하드웨어 준비 에서 노드 등록을 클릭합니다.
- 노드 화면에서 노드 등록을 클릭합니다.

그러면 노드 등록 창이 표시됩니다.



director에는 등록을 위해 노드 목록이 필요하므로 두 가지 방법 중 하나를 사용하여 제공할 수 있습니다.

1. 노드 정의 템플릿 업로드 - 이를 위해서는파일 업로드 버튼을 클릭하고 파일을 선택해야 합니다. 노드 정의 템플릿 구문에 대해서는 6.1절. "오버클라우드에 노드 등록"을 참조하십시오.

2. 수동으로 각 노드 등록 - 이를 위해서는새로 추가 를 클릭하고 노드에 대한 세부 정보를 제공해야 합니다.

수동 등록을 위해 제공해야 하는 세부 사항은 다음과 같습니다.

이름

노드의 일반 텍스트 이름입니다. *RFC3986* 예약되지 않은 문자만 사용합니다.

드라이버

사용할 전원 관리 드라이버입니다. 이 예에서는 IPMI 드라이버(**ipmi**)를 사용하지만 다른 드라이버를 사용할 수 있습니다. 사용 가능한 드라이버는 부록 B. *전원 관리 드라이버*을 참조하십시오.

IPMI IP 주소

IPMI 장치의 IP 주소입니다.

IPMI 포트

IPMI 장치에 액세스할 수 있는 포트입니다.

IPMI 사용자 이름, IPMI 암호

IPMI 사용자 이름 및 암호입니다.

아키텍처

(선택 사항) 시스템 아키텍처입니다.

CPU 수

(선택 사항) 노드에 있는 CPU 수입니다.

메모리(MB)

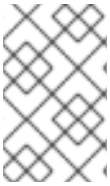
(선택 사항) 메모리 크기(MB)입니다.

디스크(GB)

(선택 사항) 하드 디스크의 크기(GB)입니다.

NIC MAC 주소

노드에 있는 네트워크 인터페이스의 MAC 주소 목록입니다. 각 시스템의 프로비저닝 NIC에는 MAC 주소만 사용합니다.



참고

UI를 사용하면 DRAC(Dell Remote Access Controller) 전원 관리를 사용하여 노드를 등록할 수도 있습니다. 이러한 노드는 **pxe_drac** 드라이버를 사용합니다. 자세한 내용은 B.2절. "DRAC(Dell Remote Access Controller)"의 내용을 참조하십시오.

노드 정보를 입력한 후 창 하단에 있는 노드 등록을 클릭합니다.

director가 노드를 등록합니다. 완료되면 UI를 사용하여 노드에서 인트로스펙션을 수행할 수 있습니다.

7.5. 웹 UI에서 노드의 하드웨어 검사

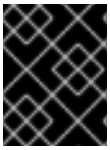
director UI는 각 노드에서 인트로스펙션 프로세스를 실행할 수 있습니다. 이 프로세스에서는 각 노드가 PXE를 통해 인트로스펙션 에이전트를 부팅합니다. 이 에이전트는 노드에서 하드웨어 데이터를 수집하고 director로 다시 보냅니다. 그러면 director가 이 인트로스펙션 데이터를 director에서 실행 중인 OpenStack Object Storage(swift) 서비스에 저장합니다. director는 프로필 태그, 벤치마킹, 수동 루트 디스크 할당과 같은 다양한 목적으로 하드웨어 정보를 사용합니다.

참고

정책 파일을 생성하여 인트로스펙션 직후 노드를 프로필에 자동으로 태그할 수도 있습니다. 정책 파일 생성 및 인트로스펙션 프로세스에 해당 파일을 포함하는 방법에 대한 자세한 내용은 부록 E. *프로파일 자동 태그*을 참조하십시오. 또는 UI를 통해 프로필에 노드를 태그할 수 있습니다. 노드를 수동으로 태그하는 방법에 대한 자세한 내용은 7.9절. "웹 UI에서 역할에 노드 할당"을 참조하십시오.

인트로스펙션 프로세스를 시작하려면 다음을 수행합니다.

1. 노드 화면으로 이동합니다.
2. 인트로스펙션을 수행하려는 노드를 모두 선택합니다.
3. 인트로스펙션을 클릭합니다.



중요

이 프로세스가 완료되었는지 확인합니다. 베어 메탈 노드의 경우 이 프로세스는 일반적으로 15분 정도 걸립니다.

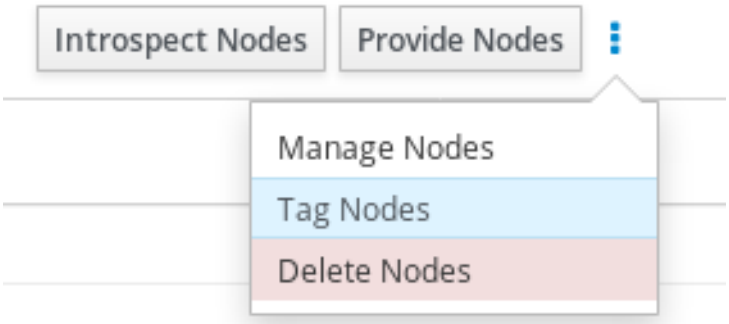
인트로스펙션 프로세스가 완료되면 프로비저닝 상태가 **manageable** 로 설정된 노드를 모두 선택한 다음 노드 제공 버튼을 클릭합니다. 프로비저닝 상태가 **available** 로 변경될 때까지 기다립니다.

이제 노드를 태그하고 프로비저닝할 준비가 되었습니다.

7.6. 웹 UI의 프로필에 노드 태그

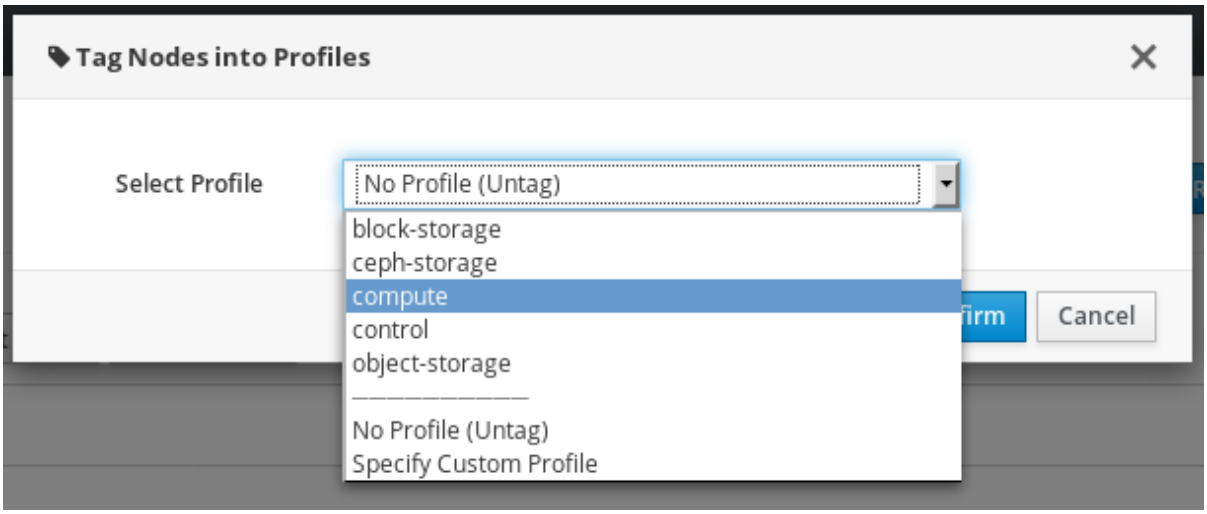
각 노드에 프로필 세트를 할당할 수 있습니다. 각 프로필은 해당 플레이버 및 역할에 해당합니다(자세한 내용은 6.5절. "프로필에 노드 태그" 참조).

노드 화면에는 태그 노드와 같은 추가 노드 관리 작업을 제공하는 추가 메뉴 토글이 포함되어 있습니다



노드 집합에 태그를 지정하려면 다음을 수행합니다.

1. 확인란을 사용하여 태그하려는 노드를 선택합니다.
2. 메뉴 토글을 클릭합니다.
3. 노드 태그 를 클릭합니다.
4. 기존 프로필을 선택합니다. 새 프로필을 생성하려면 사용자 지정 프로필 지정을 선택하고 사용자 지정 프로필에 이름을 입력합니다.



참고

사용자 지정 프로필을 생성하는 경우 새 플레이버에 프로필 태그를 할당해야 합니다. 새 플레이버 생성에 대한 자세한 내용은 6.5절. "프로필에 노드 태그"를 참조하십시오.

5. 확인 을 클릭하여 노드를 태그합니다.

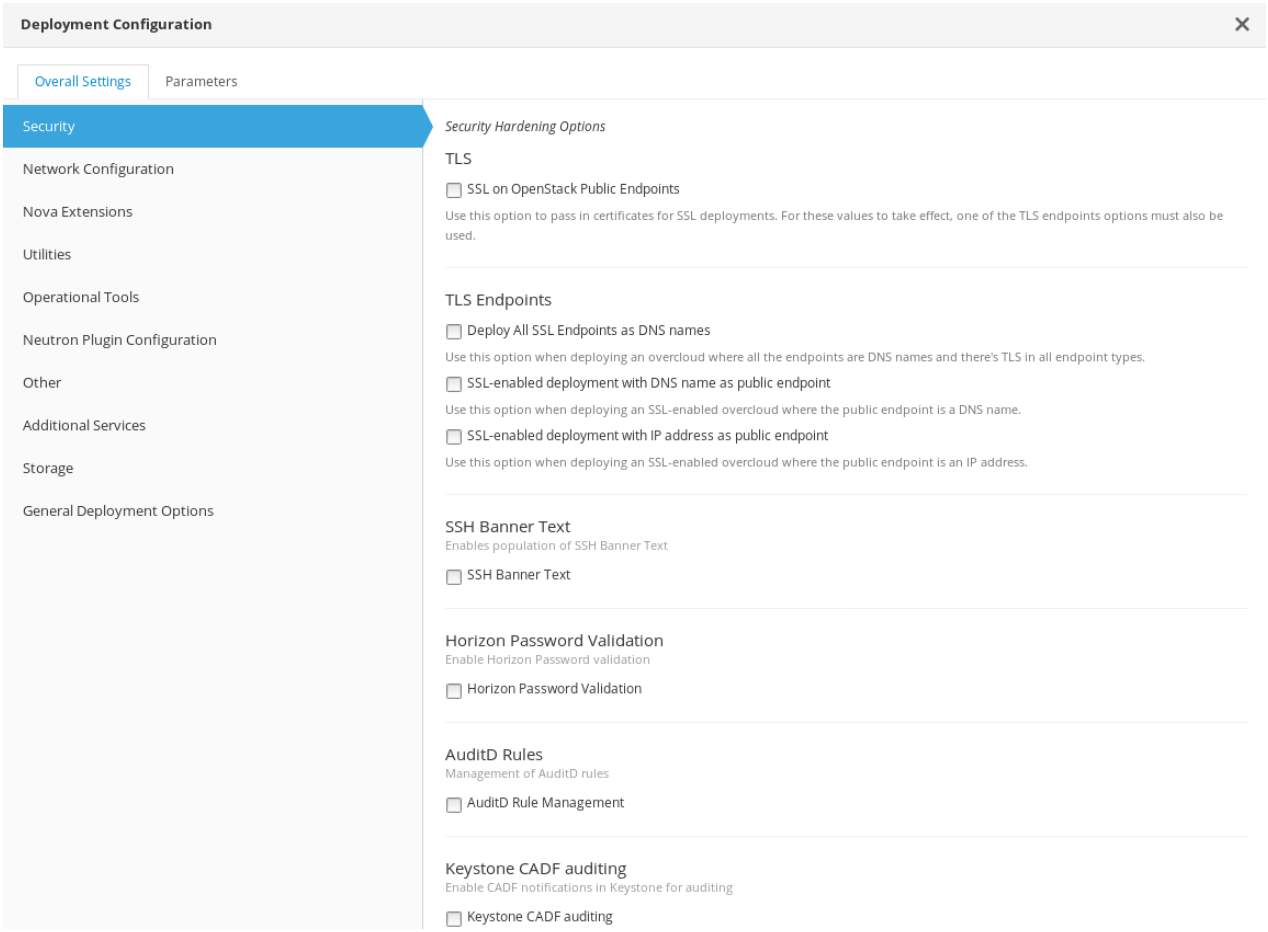
7.7. 웹 UI에서 오버클라우드 플랜 매개 변수 편집

플랜 화면에서는 업로드된 플랜을 사용자 지정할 수 있습니다.2 배포 구성 지정에서 구성 편집 링크를 클릭하여 기본 오버클라우드 구성을 수정합니다.

두 개의 기본 탭이 있는 창이 나타납니다.

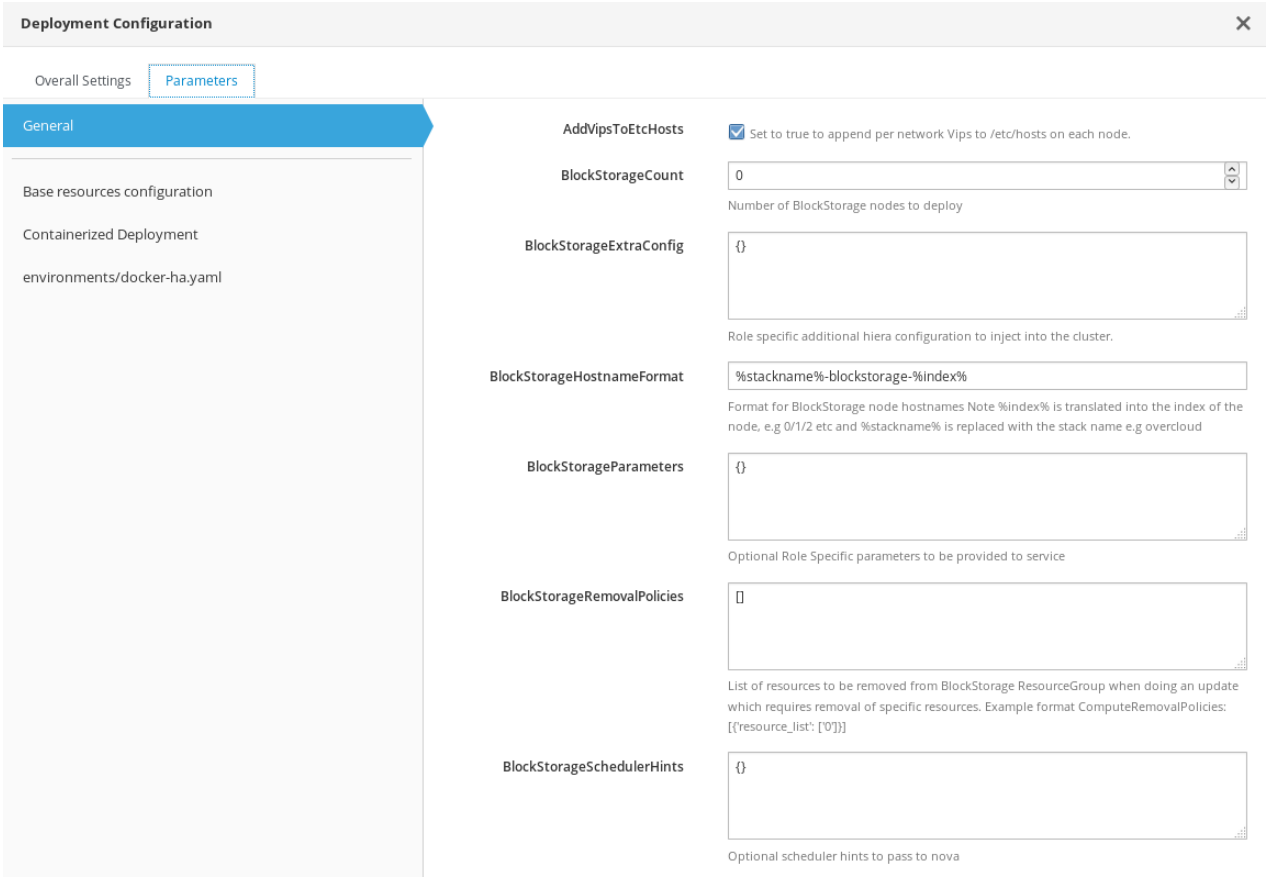
전체 설정

이렇게 하면 오버클라우드의 다른 기능을 추가할 수 있습니다. 이러한 기능은 플랜의 capabilities-map.yaml 파일에 정의되어 있으며 각 기능은 다른 환경 파일을 사용합니다. 예를 들어스토리지 에서 플랜이 environments/storage-environment.yaml 파일에 매핑되고 오버클라우드에 대한 NFS, iSCSI 또는 Ceph 설정을 구성할 수 있는 스토리지 환경을 선택할 수 있습니다.기타 탭에는 플랜에서 감지되었지만 capabilities-map.yaml 에 나열되지 않은 환경 파일이 포함되어 있습니다. 이는 플랜에 포함된 사용자 정의 환경 파일을 추가하는 데 유용합니다. 포함할 기능을 선택한 후 변경사항 저장을 클릭합니다.



매개 변수

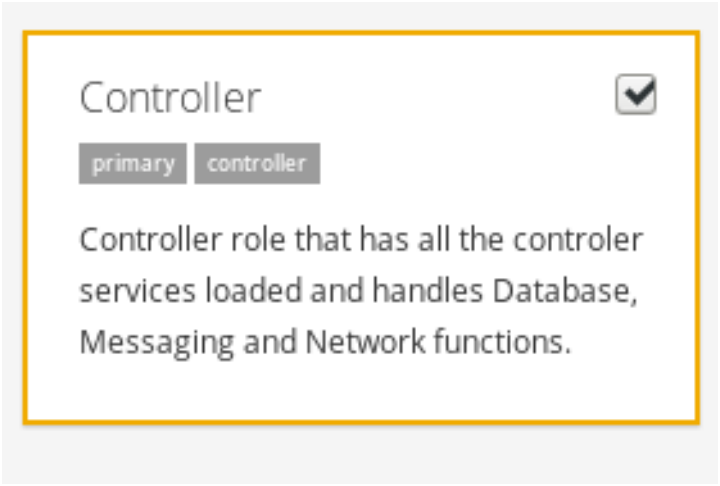
여기에는 오버클라우드에 대한 다양한 기본 수준 및 환경 파일 매개변수가 포함됩니다. 매개 변수를 수정한 후 변경사항 저장을 클릭합니다.



7.8. 웹 UI에서 역할 추가

역할 구성 및 노드 할당섹션의 오른쪽 하단에역할 관리 아이콘이 있습니다.

이 아이콘을 클릭하면 환경에 추가할 수 있는 역할을 나타내는 카드가 표시됩니다. 역할을 추가하려면 역할의 오른쪽 상단에 있는 확인란을 선택하십시오.

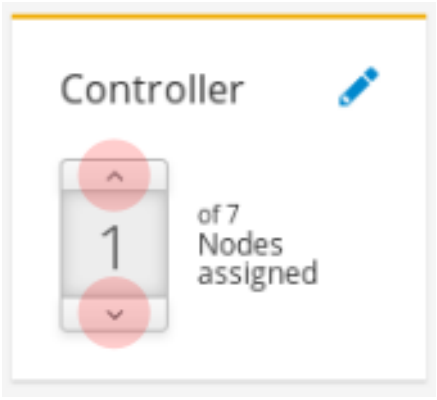


역할을 선택한 후 변경사항 저장을 클릭합니다.

7.9. 웹 UI에서 역할에 노드 할당

각 노드의 하드웨어 등록 및 검사 후 플랜의 역할에 할당합니다.

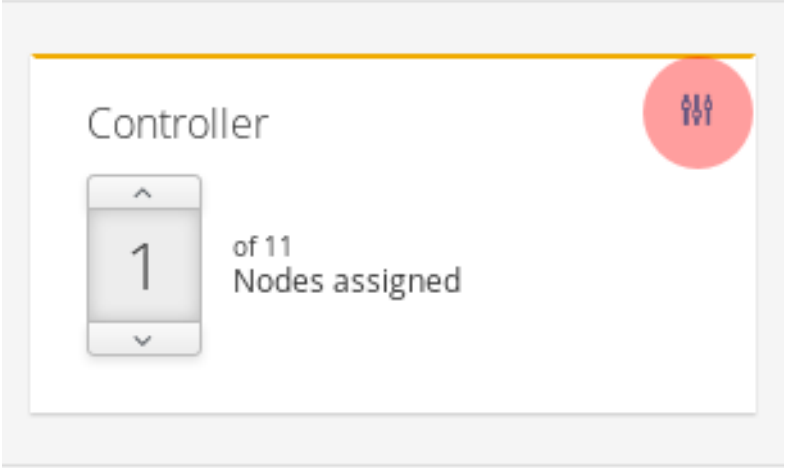
노드를 역할에 할당하려면 플랜 화면에서 3 역할 구성 및 노드 할당 섹션으로 스크롤합니다. 각 역할은 스피너 위젯을 사용하여 노드 수를 역할에 할당합니다. 역할당 사용 가능한 노드는 7.6절. "웹 UI의 프로필에 노드 태그" 에서 태그된 노드를 기반으로 합니다.



이렇게 하면 각 역할에 대해 *Count 매개 변수가 변경됩니다. 예를 들어 컨트롤러 역할의 노드 수를 3으로 변경하는 경우 ControllerCount 매개 변수를 3 으로 설정합니다. 배포 구성의 매개 변수 탭에서 개수 값을 보고 편집할 수도 있습니다. 자세한 내용은 7.7절. "웹 UI에서 오버클라우드 플랜 매개 변수 편집"를 참조하십시오.

7.10. 웹 UI에서 역할 매개 변수 편집

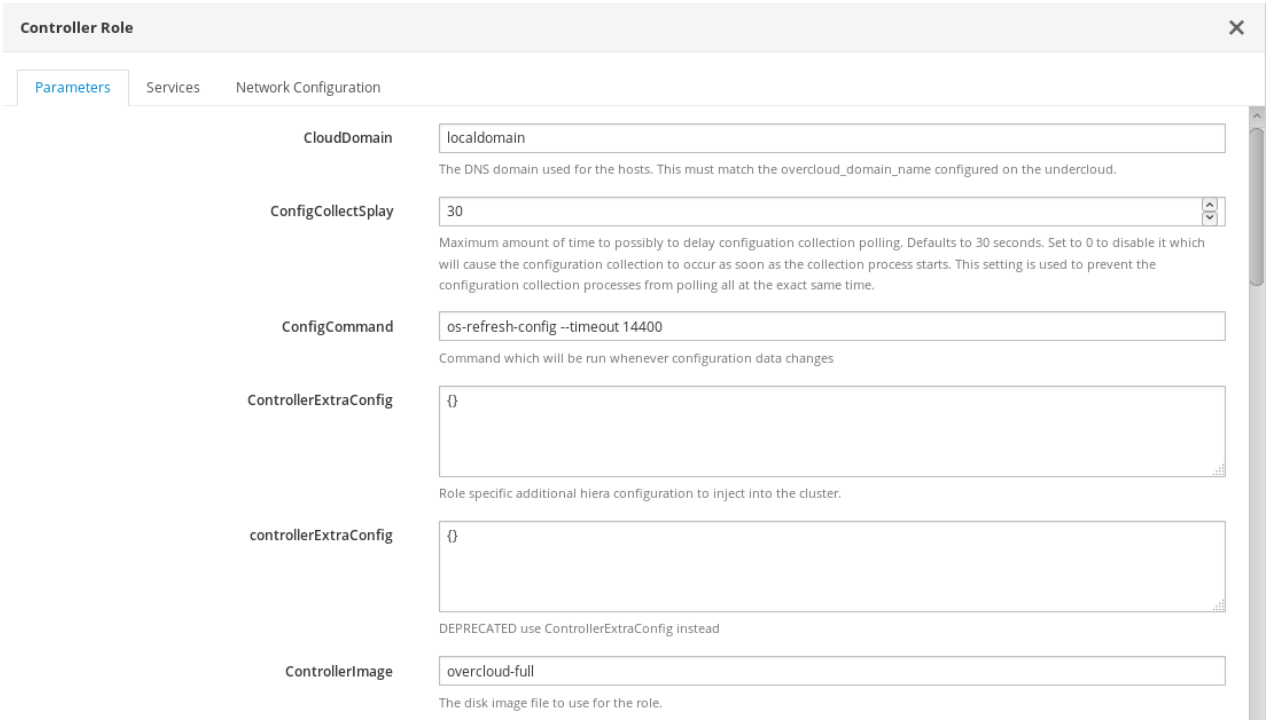
각 노드 역할은 역할별 매개 변수를 구성하는 방법을 제공합니다. 플랜 화면에서 3 역할 구성 및 노드 할당 역할로 스크롤합니다. 역할 이름 옆에 있는 역할 매개 변수 편집 아이콘을 클릭합니다.



다음 두 개의 기본 탭이 표시되는 창이 나타납니다.

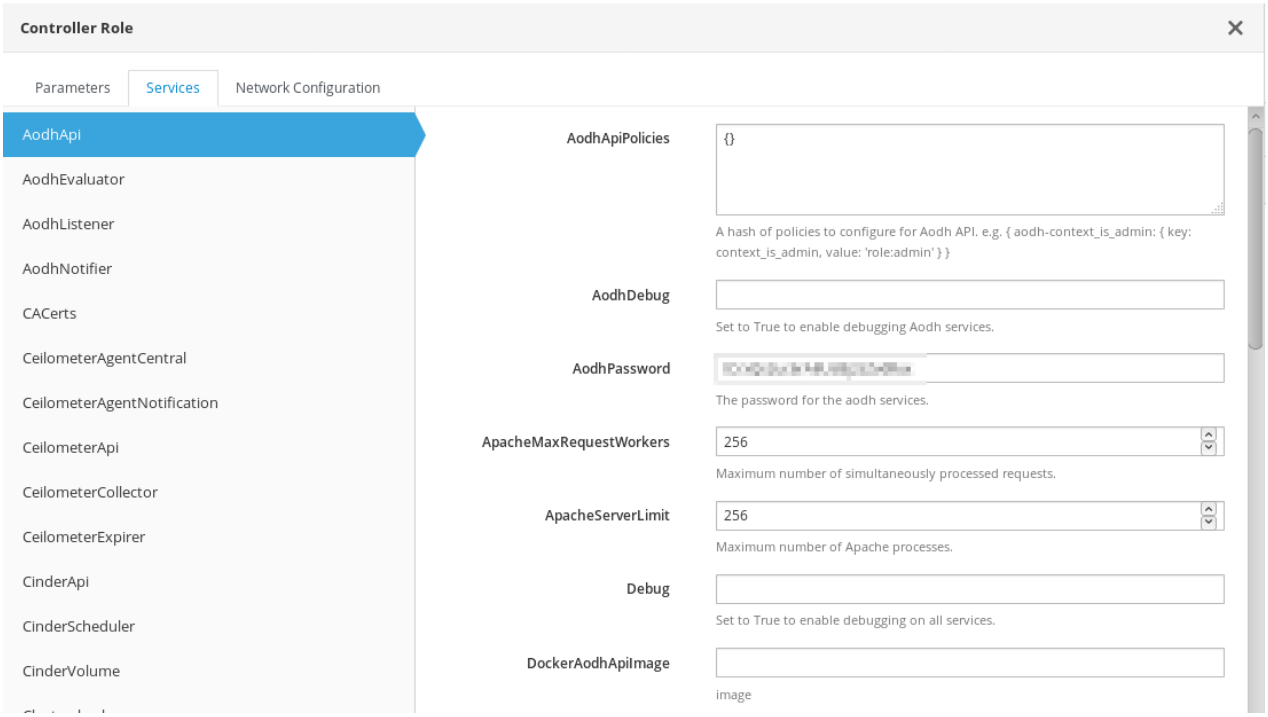
매개 변수

여기에는 다양한 역할별 매개 변수가 포함됩니다. 예를 들어 컨트롤러 역할을 편집하는 경우 OvercloudControlFlavor 매개변수를 사용하여 역할에 대한 기본 플레이버를 변경할 수 있습니다. 역할별 매개 변수를 수정한 후 변경사항 저장을 클릭합니다.



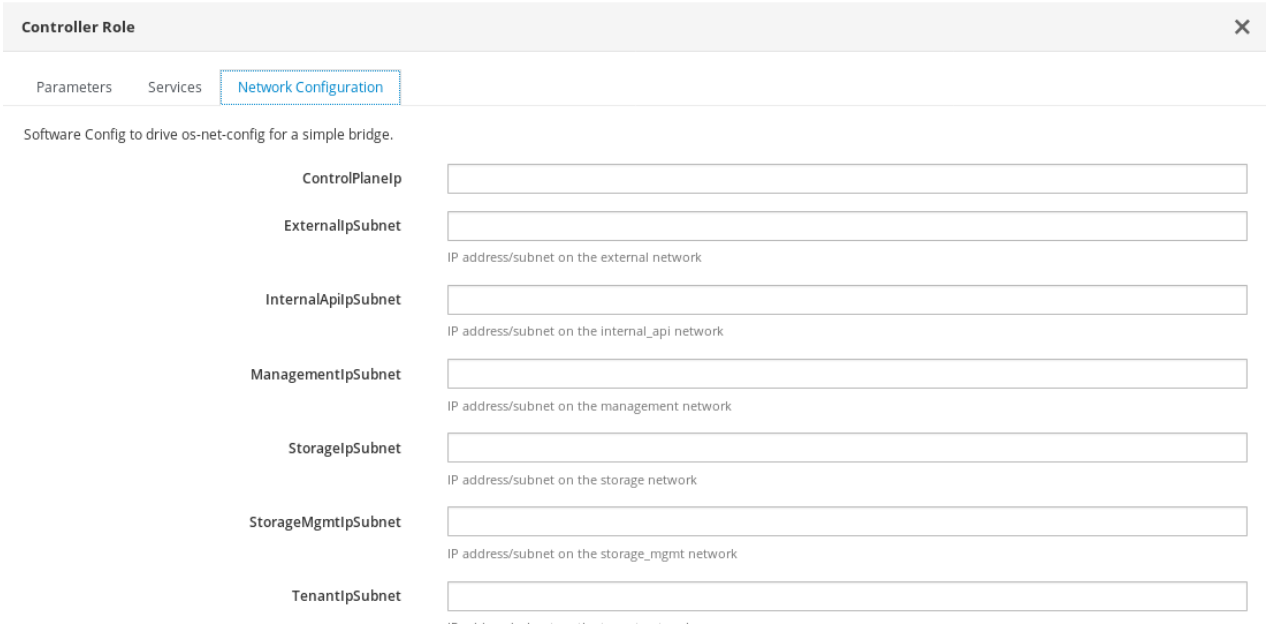
서비스

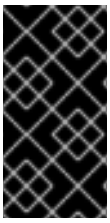
이는 선택한 역할에 대한 서비스별 매개 변수를 정의합니다. 왼쪽 패널에는 선택 및 수정한 서비스 목록이 표시됩니다. 예를 들어 시간대를 변경하려면 **OS::TripleO:Services:Timezone** 서비스를 클릭하고 **TimeZone** 매개 변수를 원하는 시간대로 변경합니다. 서비스별 매개변수를 수정한 후변경사항 저장을 클릭합니다.



네트워크 설정

이를 통해 오버클라우드의 다양한 네트워크에 대한 IP 주소 또는 서브넷 범위를 정의할 수 있습니다.





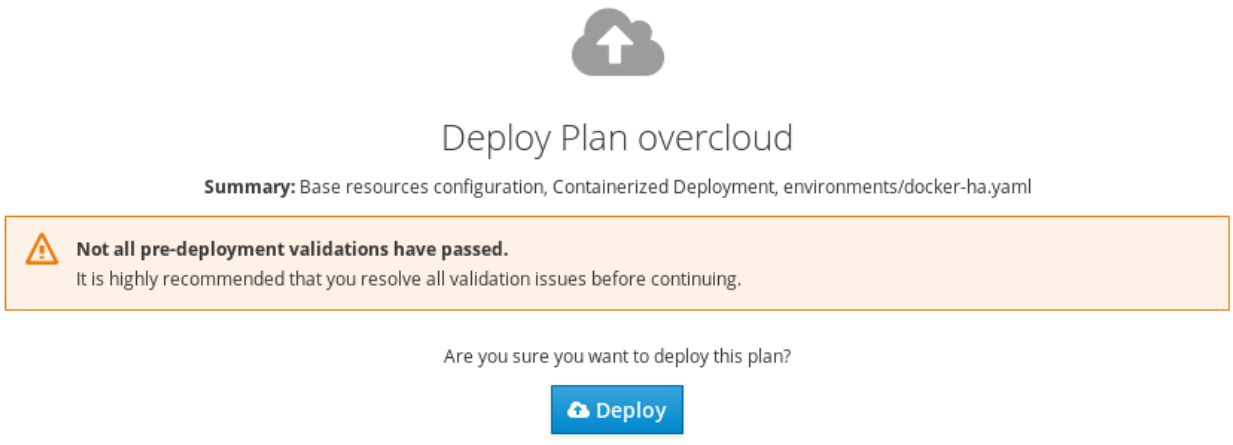
중요

역할의 서비스 매개 변수가 UI에 표시되지만 일부 서비스는 기본적으로 비활성화될 수 있습니다. 이러한 서비스는 7.7절. "웹 UI에서 오버클라우드 플랜 매개 변수 편집"의 지침을 통해 활성화할 수 있습니다. 이러한 서비스 활성화에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 *Composable Roles* 섹션을 참조하십시오.

7.11. 웹 UI에서 오버클라우드 생성 시작

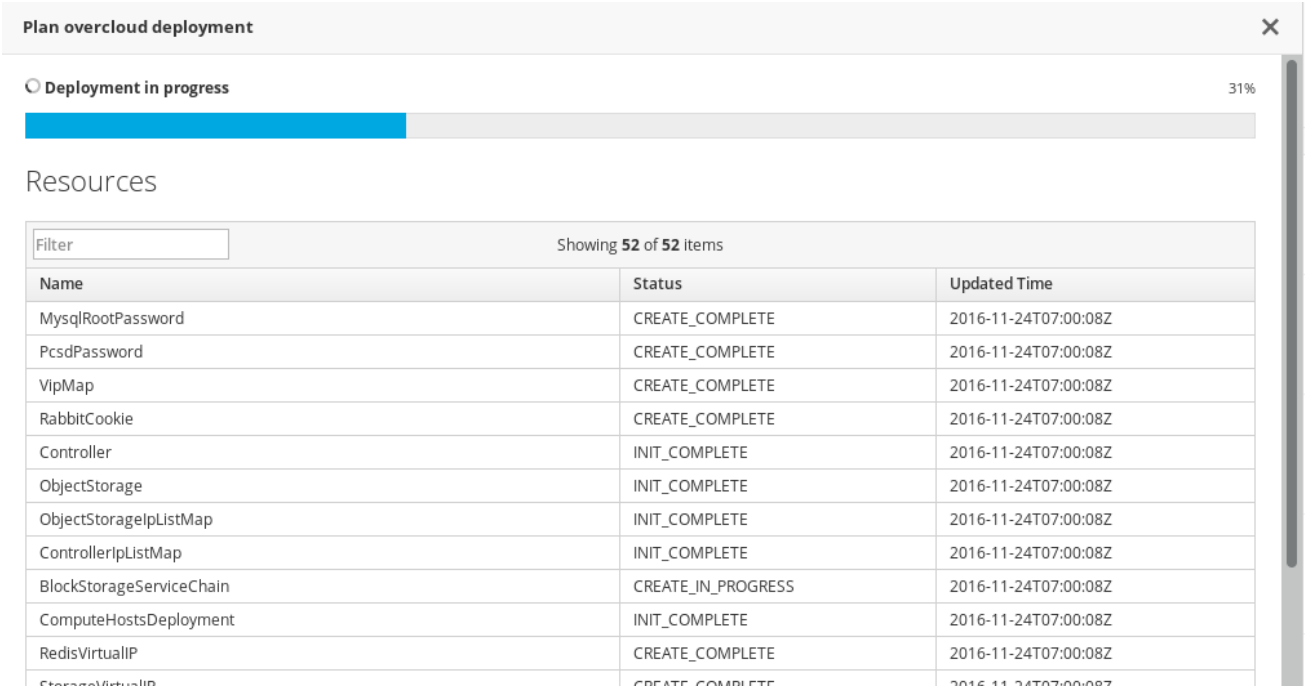
오버클라우드 플랜이 설정되면 오버클라우드 배포를 시작할 수 있습니다. 이를 위해서는 4 배포 섹션으로 스크롤하고 검증 및 배포를 클릭합니다.

언더클라우드에 대한 모든 검증을 실행하지 않았거나 통과하지 않은 경우 경고 메시지가 표시됩니다. 배포를 실행하기 전에 언더클라우드 호스트가 요구 사항을 충족하는지 확인합니다.



배포할 준비가 되었으면 배포를 클릭합니다.

UI는 오버클라우드의 생성 진행률을 정기적으로 모니터링하고 현재 진행률을 나타내는 진행률 표시줄을 표시합니다. 자세한 정보 보기 링크를 클릭하면 오버클라우드의 현재 OpenStack Orchestration 스택 로그가 표시됩니다.

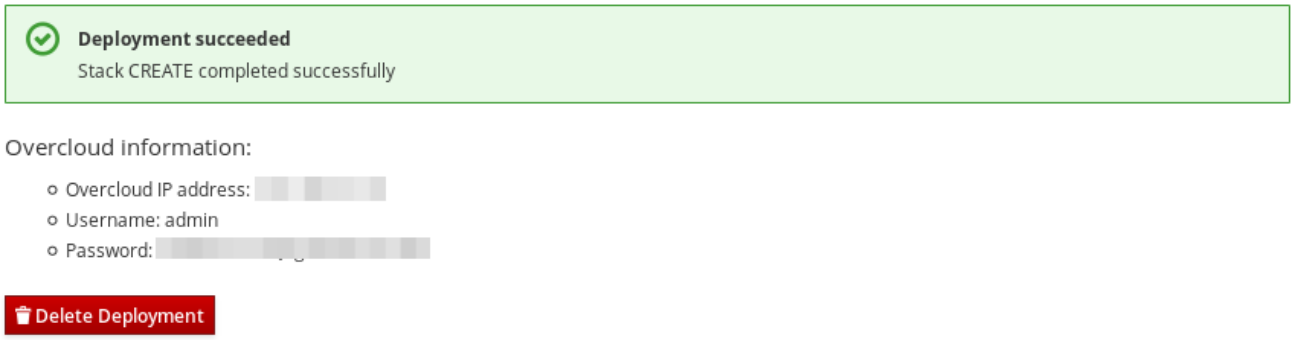


오버클라우드 배포가 완료될 때까지 기다립니다.

오버클라우드 생성 프로세스가 완료되면 4 배포 섹션에 현재 오버클라우드 상태 및 다음 세부 정보가 표시됩니다.

- IP 주소 - 오버클라우드에 액세스하는 IP 주소입니다.
- 암호 - 오버클라우드의 OpenStack **admin** 사용자의 암호입니다.

이 정보를 사용하여 오버클라우드에 액세스합니다.



7.12. 오버클라우드 생성 완료

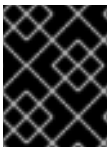
이렇게 하면 director의 UI를 통한 오버클라우드 생성이 완료됩니다. 생성 후 기능에 대해서는 9장. *오버클라우드 생성 후 작업 수행*에서 참조하십시오.

8장. 사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 구성

이 장에서는 사전 프로비저닝된 노드를 사용하여 OpenStack Platform 환경을 구성하기 위한 기본 설정 단계를 설명합니다. 이 시나리오는 여러 방식에서 표준 오버클라우드 생성 시나리오와 다릅니다.

- 외부 툴을 사용하여 노드를 프로비저닝하고 director가 오버클라우드 구성만 제어하도록 할 수 있습니다.
- director의 프로비저닝 방법에 의존하지 않고 노드를 사용할 수 있습니다. 이는 전원 관리 제어 없이 오버클라우드를 생성하거나 DHCP/PXE 부팅 제한이 있는 네트워크를 사용하는 경우 유용합니다.
- director에서 노드 관리에 OpenStack Compute(nova), OpenStack Bare Metal(ironic) 또는 OpenStack Image(glance)를 사용하지 않습니다.
- 사전 프로비저닝된 노드에서는 사용자 지정 파티션 레이아웃을 사용합니다.

이 시나리오는 사용자 지정 기능이 없는 기본 구성을 제공합니다. 하지만 Advanced Overcloud Customization 가이드의 지침에 따라 이 기본 오버클라우드에 고급 옵션을 추가하고 사양에 맞게 사용자 지정할 수 있습니다.



중요

사전 프로비저닝된 노드를 오버클라우드의 director 프로비저닝 노드와 혼합하는 것은 지원되지 않습니다.

요구 사항

- 4장. *언더클라우드 설치* 에서 생성한 director 노드
- 노드에 사용할 베어 메탈 시스템 세트. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다(Overcloud 역할에 대한 내용은 3.1절. "노드 배포 역할 계획" 참조). 또한 이러한 머신은 각 노드 유형에 설정된 요구 사항을 준수해야 합니다. 이러한 요구 사항은 2.4절. "오버클라우드 요구 사항" 에서 참조하십시오. 이러한 노드에는 Red Hat Enterprise Linux 7.5 이상을 호스트 운영 체제로 설치해야 합니다. 최신 버전을 사용하는 것이 좋습니다.
- 사전 프로비저닝된 노드를 관리하기 위한 하나의 네트워크 연결. 이 시나리오를 수행하려면 오케스트레이션 에이전트 구성을 위해 노드에 SSH 액세스가 중단되지 않도록 해야 합니다.
- 컨트롤 플레인 네트워크에 대한 하나의 네트워크 연결. 이 네트워크에는 두 가지 시나리오가 있습니다.
  - 프로비저닝 네트워크를 컨트롤 플레인으로 사용(기본 시나리오). 이 네트워크는 일반적으로 사전 프로비저닝된 노드에서 director로 연결되는 3 계층(L3) 라우팅 가능한 네트워크 연결입니다. 이 시나리오의 예는 다음 IP 주소 할당을 사용합니다.

    표 8.1. 프로비저닝 네트워크 IP 할당

노드 이름	IP 주소
Director	192.168.24.1
Controller 0	192.168.24.2
Compute 0	192.168.24.3

- 별도 네트워크 사용. director의 프로비저닝 네트워크가 라우팅 불가능한 개인 네트워크인 경우 서브넷에서 노드의 IP 주소를 정의하고 공용 API 엔드포인트를 통해 director와 통신할 수 있습니다. 이 시나리오에는 특정 주의 사항이 있습니다. 이 장은 8.6절. "오버클라우드 노드에 별도의 네트워크 사용" 의 뒷부분에서 조사합니다.

- 이 예에 있는 다른 모든 네트워크 유형에서도 OpenStack 서비스에 컨트롤 플레인 네트워크를 사용합니다. 하지만 다른 네트워크 트래픽 유형에 대해 추가 네트워크를 생성할 수 있습니다.

- 노드가 Pacemaker 리소스를 사용하는 경우 서비스 사용자 **hacluster** 및 서비스 그룹 **haclient**의 UID/GID는 189여야 합니다. 이는 CVE-2018-16877 때문입니다. Pacemaker를 운영 체제와 함께 설치한 경우 설치 시 이러한 ID가 자동으로 생성됩니다. ID 값을 잘못 설정한 경우에는 OpenStack minor update / fast-forward upgrade can fail on the controller nodes at pacemaker step with "Could not evaluate: backup_cib"의 절차에 따라 ID 값을 변경합니다.

- 일부 서비스가 잘못된 IP 주소에 바인딩되고 배포 실패가 발생하지 않게 하려면 **/etc/hosts** 파일에 **node-name=127.0.0.1** 매핑이 포함되지 않도록합니다.

8.1. 노드 구성을 위한 사용자 생성

이 프로세스의 후반 단계에서 director는 오버클라우드 노드에 **stack** 사용자로 SSH 액세스가 필요합니다.

1. 각 오버클라우드 노드에서 **stack** 이라는 사용자를 생성하고 각 노드에 암호를 설정합니다. 예를 들어 컨트롤러 노드에서 다음을 사용합니다.

```
[root@controller-0 ~]# useradd stack
[root@controller-0 ~]# passwd stack  # specify a password
```

2. **sudo** 사용 시 이 사용자가 암호를 요구하지 않도록 합니다.

```
[root@controller-0 ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a
/etc/sudoers.d/stack
[root@controller-0 ~]# chmod 0440 /etc/sudoers.d/stack
```

3. 사전 프로비저닝된 모든 노드에 **stack** 사용자를 생성 및 구성했으면 director 노드에서 각 오버클라우드 노드로 **stack** 사용자의 공용 SSH 키를 복사합니다. 예를 들어 director의 공용 SSH 키를 컨트롤러 노드에 복사하려면 다음을 수행합니다.

```
[stack@director ~]$ ssh-copy-id stack@192.168.24.2
```

8.2. 노드의 운영 체제 등록

각 노드는 Red Hat 서브스크립션에 액세스할 수 있어야 합니다.

중요

독립 실행형 Ceph 노드는 예외이며 Red Hat OpenStack Platform 서브스크립션이 필요하지 않습니다. 독립 실행형 Ceph 노드의 경우 director를 최신 ansible 패키지를 설치해야 합니다. 활성 Red Hat OpenStack Platform 서브스크립션이 없는 모든 Ceph 노드에서 **rhel-7-server-openstack-13-deployment-tools-rpms** 리포지토리를 활성화하여 Red Hat OpenStack Platform 호환 배포 툴을 얻는 것이 중요합니다.

다음 절차에서는 각 노드를 Red Hat Content Delivery Network에 등록하는 방법을 보여줍니다. 각 노드에서 다음 단계를 수행합니다.

1. 등록 명령을 실행하고 메시지가 표시되면 고객 포털 사용자 이름과 암호를 입력합니다.

```
[root@controller-0 ~]# sudo subscription-manager register
```

2. Red Hat OpenStack Platform 13에 대한 인타이틀먼트 풀을 검색합니다.

```
[root@controller-0 ~]# sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
```

3. 이전 단계에 있는 풀 ID를 사용하여 Red Hat OpenStack Platform 13 인타이틀먼트를 연결합니다.

```
[root@controller-0 ~]# sudo subscription-manager attach --pool=pool_id
```

4. 모든 기본 리포지토리를 비활성화합니다.

```
[root@controller-0 ~]# sudo subscription-manager repos --disable=*
```

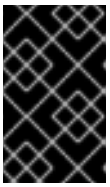
5. 필수 Red Hat Enterprise Linux 리포지토리를 활성화합니다.

   a. x86_64 시스템의 경우 다음을 실행합니다.

```
[root@controller-0 ~]# sudo subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-7-server-extras-rpms --enable=rhel-7-server-rh-common-rpms --enable=rhel-ha-for-rhel-7-server-rpms --enable=rhel-7-server-openstack-13-rpms --enable=rhel-7-server-rhceph-3-osd-rpms --enable=rhel-7-server-rhceph-3-mon-rpms --enable=rhel-7-server-rhceph-3-tools-rpms
```

   b. POWER 시스템의 경우 다음을 실행합니다.

```
[root@controller-0 ~]# sudo subscription-manager repos --enable=rhel-7-for-power-le-rpms --enable=rhel-7-server-openstack-13-for-power-le-rpms
```



중요

2.5절. “리포지토리 요구 사항”에 나열된 리포지토리만 활성화합니다. 추가 리포지토리를 사용하면 패키지와 소프트웨어가 충돌할 수 있습니다. 추가 리포지토리를 활성화하지 마십시오.

6. 시스템을 업데이트하여 기본 시스템 패키지를 최신 상태로 유지합니다.

```
[root@controller-0 ~]# sudo yum update -y
[root@controller-0 ~]# sudo reboot
```

이제 노드에서 오버클라우드를 사용할 준비가 되었습니다.

8.3. 노드에 사용자 에이전트 설치

사전 프로비저닝된 각 노드는 OpenStack Orchestration(heat) 에이전트를 사용하여 director와 통신합니다. 각 노드의 에이전트는 director를 폴링하고 각 노드에 맞는 메타데이터를 가져옵니다. 이 메타데이터를 사용하면 에이전트가 각 노드를 구성할 수 있습니다.

각 노드에 오케스트레이션 에이전트의 초기 패키지를 설치합니다.

```
[root@controller-0 ~]# sudo yum -y install python-heat-agent*
```

8.4. DIRECTOR에 대한 SSL/TLS 액세스 구성

director가 SSL/TLS를 사용하는 경우 사전 프로비저닝된 노드에 director의 SSL/TLS 인증서에 서명하는 데 사용된 인증 기관 파일이 필요합니다. 자체 인증 기관을 사용하는 경우 각 오버클라우드 노드에서 다음을 수행합니다.

1. 인증 기관 파일을 사전 프로비저닝된 각 노드의 **/etc/pki/ca-trust/source/anchors/** 디렉터리에 복사합니다.
2. 각 오버클라우드 노드에서 다음 명령을 실행합니다.

```
[root@controller-0 ~]#  sudo update-ca-trust extract
```

이렇게 하면 오버클라우드 노드에서 SSL/TLS를 통해 director의 공용 API에 액세스할 수 있습니다.

8.5. 컨트롤 플레인의 네트워킹 구성

사전 프로비저닝된 오버클라우드 노드는 표준 HTTP 요청을 사용하여 director에서 메타데이터를 가져옵니다. 따라서 모든 오버클라우드 노드에 다음 중 하나에 대한 L3 액세스 권한이 필요합니다.

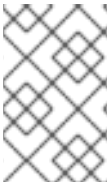
- director의 컨트롤 플레인 네트워크. **undercloud.conf** 파일의 **network_cidr** 매개변수로 정의된 서브넷입니다. 노드에는 이 서브넷에 대한 직접 액세스 권한이나 서브넷으로 라우팅 가능한 액세스 권한이 필요합니다.
- director의 공용 API 엔드포인트. **undercloud.conf** 파일의 **undercloud_public_host** 매개변수로 지정됩니다. 이 옵션은 컨트롤 플레인에 대한 L3 경로가 없거나 director를 메타데이터에 폴링할 때 SSL/TLS 통신을 사용하려는 경우 사용할 수 있습니다. 공용 API 엔드포인트를 사용하도록 오버클라우드 노드를 구성하는 추가 단계는 8.6절. "오버클라우드 노드에 별도의 네트워크 사용"을 참조하십시오.

director는 컨트롤 플레인 네트워크를 사용하여 표준 오버클라우드를 관리하고 설정합니다. 노드가 사전 프로비저닝된 오버클라우드의 경우 director가 사전 프로비저닝된 노드와 통신할 수 있도록 네트워크 구성을 일부 변경해야 할 수 있습니다.

네트워크 분리 사용

네트워크 분리를 사용하면 컨트롤 플레인을 비롯한 특정 네트워크를 사용하도록 서비스를 그룹화할 수 있습니다. Advanced Overcloud Customization 가이드에 여러 네트워크 분리 설정이 포함되어 있습니다. 또한 컨트롤 플레인에서 노드의 특정 IP 주소를 정의할 수도 있습니다. 네트워크 분리 및 예측 가능한 노드 배치 전략에 대한 자세한 내용은 *Advanced Overcloud Customization 가이드의* 다음 섹션을 참조하십시오.

- "Basic Network Isolation"
- "Controlling Node Placement"



참고

네트워크 분리를 사용하는 경우 NIC 템플릿에 언더클라우드 액세스에 사용된 NIC가 포함되지 않도록 합니다. 이러한 템플릿은 NIC를 재구성할 수 있으므로 배포 중에 연결 및 설정 문제가 발생할 수 있습니다.

IP 주소 할당

네트워크 분리를 사용하지 않는 경우 단일 컨트롤 플레인 네트워크를 사용하여 모든 서비스를 관리할 수 있습니다. 이렇게 하려면 컨트롤 플레인 네트워크 범위 내에 있는 IP 주소를 사용하도록 각 노드에서 컨트롤 플레인 NIC를 수동으로 설정해야 합니다. director의 프로비저닝 네트워크를 컨트롤 플레인으로 사용하는 경우 선택한 오버클라우드 IP 주소가 프로비저닝(**dhcp_start** 및 **dhcp_end**)과 인트로스펙션(**inspection_iprange**)에 대한 DHCP 범위 외부에 있는지 확인합니다.

표준 오버클라우드 생성 중에 director는 OpenStack Networking(neutron) 포트를 생성하여 프로비저닝/컨트롤 플레인 네트워크의 오버클라우드 노드에 IP 주소를 자동으로 할당합니다. 그러나 이로 인해 director에서 다른 IP 주소를 각 노드에 대해 수동으로 구성된 주소에 할당할 수 있습니다. 이 경우 예측 가능한 IP 주소 할당 방법을 사용하여 director에서 컨트롤 플레인에 사전 프로비저닝된 IP 할당을 사용하도록 강제 적용합니다.

예측 가능한 IP 전략의 예는 다음 IP가 할당된 환경 파일(**ctlplane-assignments.yaml**)을 사용하는 것입니다.

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml

parameter_defaults:
  DeployedServerPortMap:
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.2
      subnets:
        - cidr: 24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.3
      subnets:
        - cidr: 24
```

이 예제에서 **OS::TripleO::DeployedServer::ControlPlanePort** 리소스는 매개변수 집합을 director에 전달하고, 사전 프로비저닝된 노드의 IP 할당을 정의합니다. **DeployedServerPortMap** 매개변수는 각 오버클라우드 노드에 해당하는 IP 주소와 서브넷 CIDR을 정의합니다. 매핑은 다음을 정의합니다.

1. 할당 이름 - **<node_hostname>-<network>** 포맷을 따릅니다. 여기서 **<node_hostname>** 값은 노드의 짧은 호스트 이름으로 **<network>**는 네트워크의 소문자를 사용한 이름입니다. 예를 들어 **controller-0.example.com**의 **controller-0-ctlplane**와 **compute-0.example.com**의 **compute-0-ctlplane**입니다.

2. IP 할당 - 다음 매개변수 패턴을 사용합니다.

    - **fixed_ips/ip_address** - 컨트롤 플레인의 고정 IP 주소를 정의합니다. 목록에 여러 개의 **ip_address** 매개변수를 사용하여 여러 IP 주소를 정의합니다.

    - **subnets/cidr** - 서브넷의 CIDR 값을 정의합니다.

이 장의 이후 단계에서는 결과 환경 파일(**ctlplane-assignments.yaml**)을 **openstack overcloud deploy** 명령의 일부로 사용합니다.

8.6. 오버클라우드 노드에 별도의 네트워크 사용

기본적으로 director는 프로비저닝 네트워크를 오버클라우드 컨트롤 플레인으로 사용합니다. 하지만 이 네트워크가 분리되어 라우팅이 불가능한 경우 구성 중에 노드에서 director의 내부 API와 통신할 수 없습니다. 이 경우 노드에 대해 별도의 네트워크를 정의하고, 공용 API로 director와 통신하도록 구성해야 할 수 있습

니다.

이 시나리오에는 다음과 같은 여러 요구 사항이 있습니다.

- 오버클라우드 노드는 8.5절. "컨트롤 플레인의 네트워킹 구성"의 기본 네트워크 설정을 수용해야 합니다.
- director에서 공용 API 엔드포인트 사용에 대해 SSL/TLS를 활성화해야 합니다. 자세한 내용은 4.9절. "director 설정 매개변수" 및 부록 A. *SSL/TLS 인증서 구성*을 참조하십시오.
- director에 대해 FQDN(정규화된 도메인 이름)을 정의해야 합니다. 이 FQDN은 director의 라우팅 가능 IP 주소로 확인되어야 합니다. **undercloud.conf** 파일에서 **undercloud_public_host** 매개변수를 사용하여 이 FQDN을 설정합니다.

이 섹션의 예에서는 기본 시나리오와 다른 IP 주소 할당을 사용합니다.

표 8.2. 프로비저닝 네트워크 IP 할당

노드 이름	IP 주소 또는 FQDN
Director(내부 API)	192.168.24.1(프로비저닝 네트워크 및 컨트롤 플레인)
Director(공용 API)	10.1.1.1 / director.example.com
오버클라우드 가상 IP	192.168.100.1
Controller 0	192.168.100.2
Compute 0	192.168.100.3

다음 섹션에서는 오버클라우드 노드에 별도의 네트워크가 필요한 경우 추가 설정 방법을 설명합니다.

오케스트레이션 구성

언더클라우드에서 SSL/TLS 통신을 활성화하면 director는 대부분의 서비스에 대해 공용 API 엔드포인트를 제공합니다. 그러나 OpenStack Orchestration(heat)은 내부 엔드포인트를 메타데이터의 기본 공급자로 사용합니다. 따라서 오버클라우드 노드를 공용 끝점의 OpenStack Orchestration에 액세스할 수 있도록 언더클라우드에 약간의 수정이 필요합니다. 이 수정에는 director에서 일부 Puppet hieradata를 변경하는 작업이 포함됩니다.

undercloud.conf 의 **hieradata_override** 를 사용하면 언더클라우드 구성에 대해 추가 Puppet hieradata를 지정할 수 있습니다. 다음 단계를 사용하여 OpenStack Orchestration과 관련된 hieradata를 수정합니다.

1. **hieradata_override** 파일을 아직 사용하지 않는 경우 새 파일을 생성합니다. 이 예에서는 **/home/stack/hieradata.yaml** 에 있는 하나를 사용합니다.
2. **/home/stack/hieradata.yaml** 에 다음 hieradata를 포함합니다.

```
heat_clients_endpoint_type: public
heat::engine::default_deployment_signal_transport: TEMP_URL_SIGNAL
```

이렇게 하면 엔드포인트 유형이 기본 **internal** 에서 **public** 으로 변경되고 OpenStack Object Storage(swift)에서 TempURLs를 사용하도록 신호 방법이 변경됩니다.

3. **undercloud.conf** 에서 **hieradata_override** 매개변수를 hieradata 파일의 경로로 설정합니다.

```
hieradata_override = /home/stack/hieradata.yaml
```

4. **openstack undercloud install** 명령을 다시 실행하여 새 구성 옵션을 구현합니다.

이렇게 하면 오케스트레이션 메타데이터 서버가 director의 공용 API에서 URL을 사용하도록 전환합니다.

IP 주소 할당

IP 할당 방법은 8.5절. "컨트롤 플레인의 네트워킹 구성"과 비슷합니다. 그러나 컨트롤 플레인은 배포된 서버에서 라우팅되지 않으므로 **DeployedServerPortMap** 매개변수를 사용하여 컨트롤 플레인에 액세스할 가상 IP 주소를 포함하여 선택한 오버클라우드 노드 서브넷에서 IP 주소를 할당합니다. 다음은 8.5절. "컨트롤 플레인의 네트워킹 구성" 의 **ctlplane-assignments.yaml** 환경 파일의 수정된 버전으로 이 네트워크 아키텍처를 지원합니다.

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::ControlPlaneVipPort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml (1)

parameter_defaults:
  NeutronPublicInterface: eth1
  EC2MetadataIp: 192.168.100.1 (2)
  ControlPlaneDefaultRoute: 192.168.100.1
  DeployedServerPortMap:
    control_virtual_ip:
      fixed_ips:
        - ip_address: 192.168.100.1
      subnets:
        - cidr: 24
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.2
      subnets:
        - cidr: 24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.3
      subnets:
        - cidr: 24
```

(1) **RedisVipPort** 리소스는 **network/ports/noop.yaml**에 매핑됩니다. 이 매핑은 기본 Redis VIP 주소가 컨트롤 플레인에서 제공되기 때문입니다. 이 경우 **noop**를 사용하여 이 컨트롤 플레인 매핑을 비활성화합니다.

(2) **EC2MetadataIp** 및 **ControlPlaneDefaultRoute** 매개변수는 컨트롤 플레인 가상 IP 주소 값으로 설정됩니다. 기본 NIC 구성 템플릿에는 이러한 매개변수가 필요하므로 ping할 수 있는 IP 주소를 사용하여 배포 중에 수행된 검증을 전달하도록 설정해야 합니다. 또는 이러한 매개변수가 필요하지 않도록 NIC 구성을 사용자 지정합니다.

8.7. 사전 프로비저닝된 노드에 대해 CEPH STORAGE 설정

이미 배포된 **ceph-ansible** 및 서버를 사용하는 경우 배포 전에 다음과 같은 명령을 실행해야 합니다.

```
export OVERCLOUD_HOSTS="192.168.1.8 192.168.1.42"

bash /usr/share/openstack-tripleo-heat-templates/deployed-server/scripts/enable-ssh-admin.sh
```

예제 **export** 명령을 사용하여 **OVERCLOUD_HOSTS** 변수를 Ceph 클라이언트로 사용하기 위한 오버클라우드 호스트의 IP 주소로 설정합니다(예: Compute, Block Storage, Image, File System, Telemetry 서비스 등). **enable-ssh-admin.sh** 스크립트는 Ansible이 Ceph 클라이언트를 설정하는 데 사용하는 오버클라우드 노드에 사용자를 구성합니다.

8.8. 사전 프로비저닝된 노드를 사용하여 오버클라우드 생성

오버클라우드 배포에서는 6.11절. "CLI 툴을 사용하여 오버클라우드 생성"의 표준 CLI 메서드를 사용합니다. 사전 프로비저닝된 노드의 경우 배포 명령에 코어 Heat 템플릿 컬렉션의 일부 추가 옵션 및 환경 파일이 필요합니다.

- **--disable-validations** - 사전 프로비저닝된 인프라에 사용되지 않는 서비스에 대한 기본 CLI 검증을 비활성화합니다. 그러지 않으면 배포에 실패합니다.
- **environments/deployed-server-environment.yaml** - 사전 프로비저닝된 인프라를 생성 및 구성하는 기본 환경 파일입니다. 이 환경 파일은 **OS::Nova::Server** 리소스를 **OS::Heat::DeployedServer** 리소스로 대체합니다.
- **environments/deployed-server-bootstrap-environment-rhel.yaml** - 사전 프로비저닝된 서버에서 부트스트랩 스크립트를 실행할 환경 파일입니다. 이 스크립트는 추가 패키지를 설치하고 오버클라우드 노드에 기본 설정을 제공합니다.
- **environments/deployed-server-pacemaker-environment.yaml** - 사전 프로비저닝된 컨트롤러 노드의 Pacemaker 구성에 대한 환경 파일입니다. 이 파일에 등록된 리소스에 대한 네임스페이스는 **deployed-server/deployed-server-roles-data.yaml**의 컨트롤러 역할 이름을 사용하며, 기본값은 **ControllerDeployedServer**입니다.
- **deployed-server/deployed-server-roles-data.yaml** - 사용자 정의 역할에 대한 예제 파일입니다. 이 파일은 기본 **roles_data.yaml** 을 복제하지만 **disable_constraints**도 포함합니다. 각 역할에 대한 **true** 매개변수입니다. 이 매개변수는 생성된 역할 템플릿에서 오케스트레이션 제약 조건을 비활성화합니다. 이러한 제약 조건은 사전 프로비저닝된 인프라에 사용되지 않는 서비스에 적용됩니다. 자체 사용자 지정 역할 파일을 사용하는 경우 **disable_constraints**를 포함해야 합니다. 각 역할이 있는 **true** 매개변수입니다. 예를 들면 다음과 같습니다.

```
- name: ControllerDeployedServer
  disable_constraints: True
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    ...
```

다음은 사전 프로비저닝된 아키텍처와 관련된 환경 파일을 사용한 오버클라우드 배포 명령의 예입니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy \
  [other arguments] \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-bootstrap-
environment-rhel.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-pacemaker-
environment.yaml \
  -r /usr/share/openstack-tripleo-heat-templates/deployed-server/deployed-server-roles-data.yaml
```

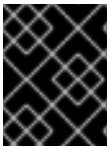
이렇게 하면 오버클라우드 구성이 시작됩니다. 그러나 오버클라우드 노드 리소스가 **CREATE_IN_PROGRESS** 단계에 진입하면 배포 스택이 일시 중지됩니다.

```
2017-01-14 13:25:13Z [overcloud.Compute.0.Compute]: CREATE_IN_PROGRESS  state changed
2017-01-14 13:25:14Z [overcloud.Controller.0.Controller]: CREATE_IN_PROGRESS  state changed
```

이 일시 중지는 director가 오버클라우드 노드에서 오케스트레이션 에이전트를 대기하여 메타데이터 서버를 폴링하기 때문입니다. 다음 섹션에서는 메타데이터 서버 폴링을 시작하도록 노드를 구성하는 방법을 보여줍니다.

8.9. 메타데이터 서버 폴링

배포가 진행 중이지만 **CREATE_IN_PROGRESS** 단계에서 일시 중지되었습니다. 다음 단계는 director의 메타데이터 서버를 폴링하도록 오버클라우드 노드에서 오케스트레이션 에이전트를 구성하는 것입니다. 이 작업을 수행하는 방법에는 두 가지가 있습니다.



중요

초기 배포에는 자동 구성만 사용합니다. 노드를 확장하는 경우 자동 구성을 사용하지 마십시오.

자동 설정

director의 코어 Heat 템플릿 컬렉션에는 오버클라우드 노드에서 Heat 에이전트의 자동 구성을 수행하는 스크립트가 포함되어 있습니다. 스크립트를 사용하려면 **stackrc** 파일을 **stack** 사용자로 가져와 director로 인증하고 오케스트레이션 서비스를 쿼리해야 합니다.

```
[stack@director ~]$ source ~/stackrc
```

또한 이 스크립트를 사용하려면 노드 역할 및 해당 IP 주소를 정의하는 몇 가지 추가 환경 변수도 필요합니다. 이러한 환경 변수는 다음과 같습니다.

OVERCLOUD_ROLES
: 구성할 역할 목록입니다. 이러한 역할은 역할 데이터 파일에 정의된 역할과 관련이 있습니다.

[ROLE]_hosts
: 각 역할에는 역할의 노드에 대해 공백으로 구분된 IP 주소 목록이 있는 환경 변수가 필요합니다.

다음 명령은 이러한 환경 변수를 설정하는 방법을 보여줍니다.

```
(undercloud) $ export OVERCLOUD_ROLES="ControllerDeployedServer ComputeDeployedServer"
(undercloud) $ export ControllerDeployedServer_hosts="192.168.100.2"
(undercloud) $ export ComputeDeployedServer_hosts="192.168.100.3"
```

스크립트를 실행하여 각 오버클라우드 노드에 오케스트레이션 에이전트를 구성합니다.

```
(undercloud) $ /usr/share/openstack-tripleo-heat-templates/deployed-server/scripts/get-occ-config.sh
```

참고

이 스크립트는 스크립트를 실행하는 동일한 사용자를 사용하여 SSH를 통해 사전 프로비저닝된 노드에 액세스합니다. 이 경우 스크립트는 **stack** 사용자로 인증합니다.

이 스크립트는 다음을 수행합니다.

- director의 오케스트레이션 서비스에서 각 노드의 메타데이터 URL을 쿼리합니다.
- 노드에 액세스하여 각 노드에서 에이전트를 특정 메타데이터 URL로 구성합니다.
- 오케스트레이션 에이전트 서비스를 다시 시작합니다.

스크립트가 완료되면 오버클라우드 노드에서 director에서 오케스트레이션 서비스 폴링을 시작합니다. 스택 배포는 계속됩니다.

수동 구성

사전 프로비저닝된 노드에서 오케스트레이션 에이전트를 수동으로 구성하려는 경우 다음 명령을 사용하여 각 노드의 메타데이터 URL에 대해 director에서 오케스트레이션 서비스를 쿼리합니다.

```
[stack@director ~]$ source ~/stackrc
(undercloud) $ for STACK in $(openstack stack resource list -n5 --filter name=deployed-server -c
stack_name -f value overcloud) ; do STACKID=$(echo $STACK | cut -d '-' -f2,4 --output-delimiter " ")
; echo "== Metadata URL for $STACKID ==" ; openstack stack resource metadata $STACK
deployed-server | jq -r '["os-collect-config"].request.metadata_url' ; echo ; done
```

그러면 각 노드의 스택 이름 및 메타데이터 URL이 표시됩니다.

```
== Metadata URL for ControllerDeployedServer 0 ==
http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-edServer-
ts6lr4tm5p44-deployed-server-td42md2tap4g/43d302fa-d4c2-40df-b3ac-624d6075ef27?
temp_url_sig=58313e577a93de8f8d2367f8ce92dd7be7aac3a1&temp_url_expires=2147483586

== Metadata URL for ComputeDeployedServer 0 ==
http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-edServer-
wdpk7upmz3eh-deployed-server-ghv7ptfikz2j/0a43e94b-fe02-427b-9bfe-71d2b7bb3126?
temp_url_sig=8a50d8ed6502969f0063e79bb32592f4203a136e&temp_url_expires=2147483586
```

각 오버클라우드 노드에서 다음을 수행합니다.

1. 기존 **os-collect-config.conf** 템플릿을 제거합니다. 이렇게 하면 에이전트에서 수동 변경 사항을 재정의하지 않습니다.

   ```
   $ sudo /bin/rm -f /usr/libexec/os-apply-config/templates/etc/os-collect-config.conf
   ```

2. 해당 메타데이터 URL을 사용하도록 **/etc/os-collect-config.conf** 파일을 구성합니다. 예를 들어 컨트롤러 노드는 다음을 사용합니다.

```
[DEFAULT]
collectors=request
command=os-refresh-config
polling_interval=30

[request]
metadata_url=http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-edServer-ts6lr4tm5p44-deployed-server-td42md2tap4g/43d302fa-d4c2-40df-b3ac-624d6075ef27?temp_url_sig=58313e577a93de8f8d2367f8ce92dd7be7aac3a1&temp_url_expires=2147483586
```

3. 파일을 저장합니다.

4. **os-collect-config** 서비스를 다시 시작합니다.

```
[stack@controller ~]$ sudo systemctl restart os-collect-config
```

오케스트레이션 에이전트에서 구성 및 재시작 후 director의 오케스트레이션 서비스를 오버클라우드 구성에 폴링합니다. 배포 스택은 생성을 계속하고 각 노드의 스택은 결국 **CREATE_COMPLETE** 로 변경됩니다.

8.10. 오버클라우드 생성 모니터링

오버클라우드 설정 프로세스가 시작됩니다. 이 프로세스를 완료하는 데 시간이 다소 걸립니다. 오버클라우드 생성 상태를 보려면 **stack** 사용자로 별도의 터미널을 열고 다음을 실행합니다.

```
[stack@director ~]$ source ~/stackrc
(undercloud) $ heat stack-list --show-nested
```

heat stack-list --show-nested 명령은 오버클라우드 생성의 현재 단계를 표시합니다.

8.11. 오버클라우드 액세스

director는 director 호스트에서 오버클라우드와의 상호 작용을 구성하고 인증을 지원하는 스크립트를 생성합니다. director는 **overcloudrc** 파일을 **stack** 사용자의 홈 director에 저장합니다. 이 파일을 사용하려면 다음 명령을 실행합니다.

```
(undercloud) $ source ~/overcloudrc
```

이렇게 하면 director 호스트의 CLI에서 오버클라우드와 상호 작용하는 데 필요한 환경 변수가 로드됩니다. 명령 프롬프트가 다음과 같이 변경됩니다.

```
(overcloud) $
```

director 호스트와의 상호 작용으로 돌아가려면 다음 명령을 실행합니다.

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

8.12. 사전 프로비저닝된 노드 확장

사전 프로비저닝된 노드를 확장하는 프로세스는 12장. *오버클라우드 노드 확장*의 표준 확장 절차와 유사합니다. 하지만 사전 프로비저닝된 노드가 OpenStack Bare Metal(ironic) 및 OpenStack Compute(nova)의 표준 등록 및 관리 프로세스를 사용하지 않으므로 사전 프로비저닝된 새 노드를 추가하는 프로세스는 다릅니다.

사전 프로비저닝된 노드 확장

사전 프로비저닝된 노드가 있는 오버클라우드를 확장하는 경우 각 노드에서 director의 노드 수에 해당하도록 오케스트레이션 에이전트를 구성해야 합니다.

사전 프로비저닝된 노드를 확장하는 일반 프로세스는 다음과 같습니다.

1. 요구 사항 에 따라 사전 프로비저닝된 새 노드를 준비합니다.
2. 노드를 확장합니다. 해당 지침은 12장. *오버클라우드 노드 확장*을 참조하십시오.
3. 배포 명령을 실행한 후 director가 새 노드 리소스를 생성할 때까지 기다립니다. 8.9절. "메타데이터 서버 폴링" 의 지침에 따라 director의 오케스트레이션 서버 메타데이터 URL을 폴링하도록 사전 프로비저닝된 노드를 수동으로 설정합니다.

사전 프로비저닝된 노드 축소

사전 프로비저닝된 노드가 있는 오버클라우드를 축소하는 경우 12장. *오버클라우드 노드 확장*에 표시된 축소 지침을 따릅니다.

대부분의 확장 작업에서는 **openstack overcloud node delete** 에 전달할 노드의 UUID 값을 가져와야 합니다. 해당 UUID를 가져오기 위해 특정 역할에 대한 리소스를 나열합니다.

```
$ openstack stack resource list overcloud -c physical_resource_id -c stack_name -n5 --filter
type=OS::TripleO::<RoleName>Server
```

위 명령에서 **<RoleName>** 을 축소하는 역할의 실제 이름으로 바꿉니다. 예를 들어 **ComputeDeployedServer** 역할의 경우 다음과 같습니다.

```
$ openstack stack resource list overcloud -c physical_resource_id -c stack_name -n5 --filter
type=OS::TripleO::ComputeDeployedServerServer
```

명령 출력에 있는 **stack_name** 열은 각 노드와 연결된 UUID를 확인하는 데 사용됩니다**stack_name** 에는 Heat 리소스 그룹에 있는 노드의 인덱스 정수 값이 포함됩니다. 예를 들어 다음 샘플 출력에서는 다음을 수행합니다.

```
+--------------------------------------+--------------------------------------+
| physical_resource_id                 | stack_name                           |
+--------------------------------------+--------------------------------------+
| 294d4e4d-66a6-4e4e-9a8b-             | overcloud-ComputeDeployedServer- |
| 03ec80beda41                         | no7yfgnh3z7e-1-ytfqdeclwvcg      |
| d8de016d-                            | overcloud-ComputeDeployedServer- |
| 8ff9-4f29-bc63-21884619abe5          | no7yfgnh3z7e-0-p4vb3meacxwn      |
| 8c59f7b1-2675-42a9-ae2c-             | overcloud-ComputeDeployedServer- |
| 2de4a066f2a9                         | no7yfgnh3z7e-2-mmmaayxqnf3o      |
+--------------------------------------+--------------------------------------+
```

stack_name 열의 인덱스 0, 1 또는 2는 Heat 리소스 그룹의 노드 순서에 해당합니다.
physical_resource_id 열의 해당 UUID 값을**openstack overcloud node delete** 명령에 전달합니다.

스택에서 오버클라우드 노드를 삭제한 경우 이러한 노드의 전원을 끕니다. 표준 배포에서는 director의 베어

베탈 서비스가 이 기능을 제어합니다. 그러나 프로비저닝된 노드를 사용하는 경우 이러한 노드를 수동으로 종료하거나 각 물리 시스템에 대해 전원 관리 컨트롤을 사용해야 합니다. 스택에서 노드를 삭제한 후 노드의 전원을 끄지 않으면 작동 상태로 남아 있어 오버클라우드 환경의 일부로 재연결될 수 있습니다.

삭제된 노드의 전원을 끈 후 기본 운영 체제 구성으로 다시 프로비저닝하면 이후에 오버클라우드에 연결되지 않을 수 있습니다.

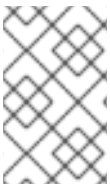
참고

먼저 새로운 기본 운영 체제로 다시 프로비저닝하지 않고 오버클라우드에서 이전에 삭제된 노드를 재사용하지 마십시오. 축소 프로세스는 오버클라우드 스택에서 노드를 삭제만 하고 패키지를 제거하지는 않습니다.

8.13. 사전 프로비저닝된 오버클라우드 삭제

사전 프로비저닝된 노드를 사용하는 전체 오버클라우드를 제거하면 표준 오버클라우드와 동일한 절차를 사용합니다. 자세한 내용은 9.12절. “오버클라우드 제거”를 참조하십시오.

오버클라우드 삭제 후 모든 노드의 전원을 끄고 기본 운영 체제 구성으로 다시 프로비저닝합니다.



참고

먼저 새로운 기본 운영 체제로 다시 프로비저닝하지 않고 오버클라우드에서 이전에 삭제된 노드를 재사용하지 마십시오. 삭제 프로세스는 오버클라우드 스택만 삭제하고 패키지를 제거하지는 않습니다.

8.14. 오버클라우드 생성 완료

이렇게 하면 사전 프로비저닝된 노드를 사용한 오버클라우드 생성이 완료됩니다. 생성 후 기능에 대해서는 9장. *오버클라우드 생성 후 작업 수행*에서 참조하십시오.

9장. 오버클라우드 생성 후 작업 수행

이 장에서는 선택한 오버클라우드를 생성한 후 수행하는 몇 가지 기능에 대해 설명합니다.

9.1. 컨테이너화된 서비스 관리

오버클라우드는 컨테이너에서 대부분의 OpenStack Platform 서비스를 실행합니다. 호스트의 개별 서비스를 제어해야 하는 경우도 있습니다. 이 섹션에서는 오버클라우드 노드에서 실행하여 컨테이너화된 서비스를 관리할 수 있는 몇 가지 일반적인 **docker** 명령을 제공합니다. **docker** 를 사용하여 컨테이너를 관리하는 방법에 대한 자세한 내용은 컨테이너 *시작하기 가이드의 "Docker 포맷된 컨테이너 사용"* 을 참조하십시오.

참고

이러한 명령을 실행하기 전에 오버클라우드 노드에 로그인하고 언더클라우드에서 이러한 명령을 실행하지 않았는지 확인합니다.

컨테이너 및 이미지 목록 표시

실행 중인 컨테이너를 나열하려면 다음을 수행합니다.

```
$ sudo docker ps
```

중지되었거나 실패한 컨테이너도 나열하려면 **--all** 옵션을 추가합니다.

```
$ sudo docker ps --all
```

컨테이너 이미지를 나열하려면 다음을 수행합니다.

```
$ sudo docker images
```

컨테이너 속성 확인

컨테이너 또는 컨테이너 이미지의 속성을 보려면 **docker inspect** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 검사하는 방법은 다음과 같습니다.

```
$ sudo docker inspect keystone
```

기본 컨테이너 작업 관리

컨테이너화된 서비스를 다시 시작하려면 **docker restart** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 다시 시작하는 방법은 다음과 같습니다.

```
$ sudo docker restart keystone
```

컨테이너화된 서비스를 중지하려면 **docker stop** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 중지하려면 다음을 실행합니다.

```
$ sudo docker stop keystone
```

중지된 컨테이너화된 서비스를 시작하려면 **docker start** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 시작하는 방법은 다음과 같습니다.

```
$ sudo docker start keystone
```

참고

컨테이너를 다시 시작한 후에 컨테이너 내의 서비스 설정 파일에 대한 모든 변경 사항을 되돌립니다. 컨테이너가 **/var/lib/config-data/puppet-generated/** 에서 노드의 로컬 파일 시스템에 있는 파일을 기반으로 서비스 구성을 다시 생성하기 때문입니다. 예를 들어 **keystone** 컨테이너 내의 **/etc/keystone/keystone.conf**를 편집하고 컨테이너를 다시 시작하는 경우 컨테이너가 노드의 로컬 파일 시스템에서 **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf**를 사용하여 구성을 다시 생성합니다. 이는 다시 시작하기 전에 컨테이너 내에 만들어진 모든 변경 사항을 덮어씁니다.

컨테이너 모니터링

컨테이너화된 서비스의 로그를 확인하려면 **docker logs** 명령을 사용합니다. 예를 들어**keystone** 컨테이너의 로그를 확인하는 방법은 다음과 같습니다.

```
$ sudo docker logs keystone
```

컨테이너 액세스

컨테이너화된 서비스 쉘에 들어가려면 **docker exec** 명령을 사용하여 **/bin/bash** 를 실행합니다. 예를 들어 **keystone** 컨테이너 쉘에 들어가는 방법은 다음과 같습니다.

```
$ sudo docker exec -it keystone /bin/bash
```

root 사용자로 **keystone** 컨테이너 쉘에 들어가려면 다음을 수행합니다.

```
$ sudo docker exec --user 0 -it <NAME OR ID> /bin/bash
```

컨테이너를 종료하려면 다음을 수행합니다.

```
# exit
```

OpenStack Platform 컨테이너화된 서비스 문제 해결에 대한 자세한 내용은 15.7.3절. “컨테이너화된 서비스 오류” 을 참조하십시오.

9.2. 오버클라우드 테넌트 네트워크 생성

오버클라우드에는 인스턴스에 대한 테넌트 네트워크가 필요합니다. **overcloud** 를 소싱하고 Neutron에서 초기 테넌트 네트워크를 생성합니다. 예를 들면 다음과 같습니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack network create default
(overcloud) $ openstack subnet create default --network default --gateway 172.20.1.1 --subnet-range
172.20.0.0/16
```

이렇게 하면 **default** 라는 기본 Neutron 네트워크가 생성됩니다. 오버클라우드는 내부 DHCP 메커니즘을 사용하여 이 네트워크의 IP 주소를 자동으로 할당합니다.

생성된 네트워크를 확인합니다.

```
(overcloud) $ openstack network list
+----------------------+------------+--------------------------------------+
| id                   | name       | subnets                              |
+----------------------+------------+--------------------------------------+
| 95fadaa1-5dda-4777... | default    | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
+----------------------+------------+--------------------------------------+
```

9.3. 오버클라우드 외부 네트워크 생성

유동 IP 주소를 인스턴스에 할당할 수 있도록 오버클라우드에서 외부 네트워크를 생성해야 합니다.

기본 VLAN 사용

다음 절차에서는 외부 네트워크에 전용 인터페이스 또는 기본 VLAN이 설정되어 있다고 가정합니다.

overcloud 를 소싱하여 Neutron에 외부 네트워크를 생성합니다. 예를 들면 다음과 같습니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-network-type flat --provider-
physical-network datacentre
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.1.51,end=10.1.1.250 --gateway 10.1.1.1 --subnet-range 10.1.1.0/24
```

이 예에서는 이름이 public 인 네트워크를 생성합니다. 오버클라우드에는 기본 유동 IP 풀에 이러한 특정 이름이 필요합니다. 이는 9.7절. "오버클라우드 검증" 의 검증 테스트에서도 중요합니다.

또한 이 명령은 네트워크를 datacentre 물리적 네트워크에 매핑합니다. 기본적으로 datacentre 는 br-ex 브리지에 매핑됩니다. 오버클라우드 생성 중에 사용자 정의 neutron 설정을 사용하지 않은 경우 이 옵션을 기본값으로 두십시오.

기본이 아닌 VLAN 사용

기본 VLAN을 사용하지 않는 경우 다음 명령을 사용하여 네트워크를 VLAN에 할당합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-network-type vlan --provider-
physical-network datacentre --provider-segment 104
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.1.51,end=10.1.1.250 --gateway 10.1.1.1 --subnet-range 10.1.1.0/24
```

provider:segmentation_id 값은 사용할 VLAN을 정의합니다. 이 경우 104를 사용할 수 있습니다.

생성된 네트워크를 확인합니다.

```
(overcloud) $ openstack network list
+----------------------+------------+--------------------------------------+
| id                   | name       | subnets                              |
+----------------------+------------+--------------------------------------+
| d474fe1f-222d-4e32... | public     | 01c5f621-1e0f-4b9d-9c30-7dc59592a52f |
+----------------------+------------+--------------------------------------+
```

9.4. 추가 유동 IP 네트워크 생성

유동 IP 네트워크는 배포 중에 추가 브릿지를 매핑한 한 **br-ex** 뿐만 아니라 모든 브릿지를 사용할 수 있습니다.

예를 들어 **br-floating** 이라는 새 브릿지를**floating** 물리 네트워크에 매핑하려면 환경 파일에서 다음을 사용합니다.

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,floating:br-floating"
```

오버클라우드 생성 후 유동 IP 네트워크를 생성합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack network create ext-net --external --provider-physical-network floating --
provider-network-type vlan --provider-segment 105
(overcloud) $ openstack subnet create ext-subnet --network ext-net --dhcp --allocation-pool
start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 --subnet-range 10.1.2.0/24
```

9.5. 오버클라우드 공급자 네트워크 생성

공급자 네트워크는 배포된 오버클라우드 외부에 있는 네트워크에 물리적으로 연결된 네트워크입니다. 기존 인프라 네트워크 또는 유동 IP 대신 라우팅을 통해 인스턴스에 직접 외부 액세스를 제공하는 네트워크일 수 있습니다.

공급자 네트워크를 생성할 때 브리지 매핑을 사용하는 물리적 네트워크와 연결합니다. 이는 유동 IP 네트워크 생성과 유사합니다. 컴퓨팅 노드가 VM 가상 네트워크 인터페이스를 연결된 네트워크 인터페이스에 직접 연결하므로 공급자 네트워크를 Controller 및 Compute 노드에 모두 추가합니다.

예를 들어 원하는 공급자 네트워크가 br-ex 브리지의 VLAN인 경우 다음 명령을 사용하여 VLAN 201에서 공급자 네트워크를 추가합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack network create provider_network --provider-physical-network datacentre --
provider-network-type vlan --provider-segment 201 --share
```

이 명령은 공유 네트워크를 생성합니다. 또한 **--share** 를 지정하지 않고 테넌트를 지정할 수도 있습니다. 해당 네트워크는 지정된 테넌트에서만 사용할 수 있습니다. 공급자 네트워크를 외부로 표시하는 경우 운영자만 해당 네트워크에 포트를 생성할 수 있습니다.

neutron에서 DHCP 서비스를 테넌트 인스턴스에 제공하도록 하려면 공급자 네트워크에 서브넷을 추가합니다.

```
(overcloud) $ openstack subnet create provider-subnet --network  provider_network --dhcp --
allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway 10.9.101.254 --subnet-range
10.9.101.0/24
```

다른 네트워크에서 공급자 네트워크를 통해 외부적으로 액세스해야 할 수 있습니다. 이 경우 다른 네트워크에서 공급자 네트워크를 통해 트래픽을 라우팅할 수 있도록 새 라우터를 생성합니다.

```
(overcloud) $ openstack router create external
(overcloud) $ openstack router set --external-gateway provider_network external
```

다른 네트워크를 이 라우터에 연결합니다. 예를 들어 **subnet1** 이라는 서브넷이 있는 경우 다음 명령을 사용하여 라우터에 연결할 수 있습니다.

-

```
(overcloud) $ openstack router add subnet external subnet1
```

그러면 **subnet1** 이 라우팅 테이블에 추가되고**subnet1** 을 사용하는 트래픽이 공급자 네트워크로 라우팅됩니다.

9.6. 기본 오버클라우드 플레이버 생성

이 가이드의 검증 단계에서는 설치에 플레이버가 포함된 것으로 간주합니다. 플레이버를 하나 이상 생성하지 않은 경우 다음 명령을 사용하여 다양한 스토리지 및 처리 기능이 있는 기본 플레이버 세트를 생성합니다.

```
$ openstack flavor create m1.tiny --ram 512 --disk 0 --vcpus 1
$ openstack flavor create m1.smaller --ram 1024 --disk 0 --vcpus 1
$ openstack flavor create m1.small --ram 2048 --disk 10 --vcpus 1
$ openstack flavor create m1.medium --ram 3072 --disk 10 --vcpus 2
$ openstack flavor create m1.large --ram 8192 --disk 10 --vcpus 4
$ openstack flavor create m1.xlarge --ram 8192 --disk 10 --vcpus 8
```

명령 옵션

RAM

ram 옵션을 사용하여 플레이버의 최대 RAM을 정의합니다.

disk

disk 옵션을 사용하여 플레이버의 하드 디스크 공간을 정의합니다.

vcpus

vcpus 옵션을 사용하여 플레이버의 가상 CPU 수량을 정의합니다.

openstack flavor create 명령을 자세히 알아보려면**$ openstack flavor create --help**를 사용합니다.

9.7. 오버클라우드 검증

오버클라우드는 OpenStack Integration Test Suite(tempest) 툴 세트를 사용하여 일련의 통합 테스트를 수행합니다. 이 섹션에서는 통합 테스트 실행 준비에 대해 설명합니다. OpenStack Integration Test Suite 사용에 대한 전체 지침은 OpenStack Integration Test Suite 가이드를 참조하십시오.

Integration Test Suite를 실행하기 전에

언더클라우드에서 이 테스트를 실행하는 경우 언더클라우드 호스트에서 오버클라우드의 내부 API 네트워크에 액세스할 수 있는지 확인합니다. 예를 들어 언더클라우드 호스트에 임시 VLAN을 추가하여 내부 API 네트워크(ID)에 액세스합니다. 201) 172.16.0.201/24 주소를 사용합니다.

```
$ source ~/stackrc
(undercloud) $ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface vlan201
type=internal
(undercloud) $ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev vlan201
```

OpenStack Integration Test Suite를 실행하기 전에 오버클라우드에 **heat_stack_owner** 역할이 있는지 확인합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack role list
+----------------------------------+------------------+
| ID                               | Name             |
```

```
+----------------------------------+------------------+
| 6226a517204846d1a26d15aae1af208f | swiftoperator    |
| 7c7eb03955e545dd86bbfeb73692738b | heat_stack_owner |
+----------------------------------+------------------+
```

역할이 없는 경우 역할을 생성합니다.

```
(overcloud) $ openstack role create heat_stack_owner
```

Integration Test Suite를 실행한 후

검증을 완료한 후 오버클라우드의 내부 API에 대한 임시 연결을 삭제합니다. 이 예제에서는 다음 명령을 사용하여 이전에 생성한 VLAN을 언더클라우드에서 삭제합니다.

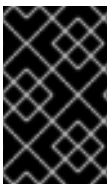
```
$ source ~/stackrc
(undercloud) $ sudo ovs-vsctl del-port vlan201
```

9.8. 오버클라우드 환경 수정

오버클라우드를 수정하여 추가 기능을 추가하거나 작동 방식을 변경할 수 있습니다. 오버클라우드를 수정하려면 사용자 정의 환경 파일 및 Heat 템플릿을 수정한 다음 초기 오버클라우드 생성 시의 **openstack overcloud deploy** 명령을 재실행합니다. 예를 들어 6.11절. "CLI 툴을 사용하여 오버클라우드 생성"을 사용하여 오버클라우드를 생성한 경우 다음 명령을 재실행합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/node-info.yaml \
  -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
```

director는 heat에서 **overcloud** 스택을 확인한 다음 스택의 각 항목을 환경 파일 및 heat 템플릿으로 업데이트합니다. 오버클라우드를 다시 생성하지 않고 기존 오버클라우드를 변경합니다.



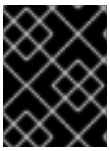
중요

사용자 지정 환경 파일에서 매개변수를 삭제해도 매개변수 값은 기본 구성으로 되돌아가지 않습니다. **/usr/share/openstack-tripleo-heat-templates**의 코어 히트 템플릿 컬렉션에서 기본값을 식별하고 사용자 지정 환경 파일에서 수동으로 값을 설정해야 합니다.

새 환경 파일을 추가하려면 **-e** 옵션을 사용하여 **openstack overcloud deploy** 명령에 추가합니다. 예를 들면 다음과 같습니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/new-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -e ~/templates/node-info.yaml \
  --ntp-server pool.ntp.org
```

여기에는 환경 파일의 새 매개변수와 리소스가 스택에 포함됩니다.



중요

오버클라우드 설정을 수동으로 수정하지 않는 것이 좋습니다. director가 나중에 이러한 수정 사항을 덮어쓸 수 있습니다.

9.9. 동적 인벤토리 스크립트 실행

director는 OpenStack Platform 환경에서 Ansible 기반 자동화를 실행할 수 있는 기능을 제공합니다. director는 **tripleo-ansible-inventory** 명령을 실행하여 환경에 있는 노드의 동적 인벤토리를 생성합니다.

절차

1. 노드의 동적 인벤토리를 보려면 **stackrc**를 소싱한 후 **tripleo-ansible-inventory** 명령을 실행합니다.

```
$ source ~/stackrc
(undercloud) $ tripleo-ansible-inventory --list
```

--list 옵션은 모든 호스트에 대한 세부 정보를 제공합니다. 동적 인벤토리가 JSON 형식으로 출력됩니다.

```
{"overcloud": {"children": ["Controller", "Compute"], "vars": {"ansible_ssh_user": "heat-admin"}}, "Controller": ["192.168.24.2"], "undercloud": {"hosts": ["localhost"], "vars": {"overcloud_horizon_url": "http://192.168.24.4:80/dashboard", "overcloud_admin_password": "abcdefghijklm12345678", "ansible_connection": "local"}}, "Compute": ["192.168.24.3"]}
```

2. 현재 환경에서 Ansible 플레이북을 실행하려면 **ansible** 명령을 실행하고 **-i** 옵션을 사용하여 동적 인벤토리 툴의 전체 경로를 포함합니다. 예를 들면 다음과 같습니다.

```
(undercloud) $ ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER OPTIONS]
```

- **[HOSTS]** 를 사용할 호스트 유형으로 변경합니다. 예를 들면 다음과 같습니다.
  - 모든 컨트롤러 노드인 경우 **controller**
  - 모든 컴퓨팅 노드인 경우 **compute**
  - 모든 오버클라우드 하위 노드인 경우 **overcloud**. 예: **controller** 및 **compute**
  - 언더클라우드인 경우 **undercloud**
  - 모든 노드인 경우 **"*"**
- **[OTHER OPTIONS]** 를 추가 Ansible 옵션으로 변경합니다. 몇 가지 유용한 옵션은 다음과 같습니다.
  - **--ssh-extra-args='-o StrictHostKeyChecking=no'**: 호스트 키 검사에서 확인을 바이패스합니다.
  - **-u [USER]**: Ansible 자동화를 실행하는 SSH 사용자를 변경합니다. 오버클라우드의 기본 SSH 사용자는 동적 인벤토리에서 **ansible_ssh_user** 매개변수를 사용하여 자동으로 정의됩니다. **-u** 옵션은 이 매개변수를 재정의합니다.

- **-m [MODULE]:** 특정 Ansible 모듈을 사용합니다. 기본값은 Linux 명령을 실행하는 **command**입니다.
- **-a [MODULE_ARGS]**: 선택한 모듈에 대한 인수를 정의합니다.

중요

오버클라우드에서 Ansible 자동화는 표준 오버클라우드 스택 외부에 속합니다. 따라서 이후 **openstack overcloud deploy** 명령을 실행하면 오버클라우드 노드에서 OpenStack Platform 서비스의 Ansible 기반 설정이 재정의될 수 있습니다.

9.10. 가상 머신을 오버클라우드로 가져오기

기존 OpenStack 환경이 있고 해당 가상 머신을 Red Hat OpenStack Platform 환경으로 마이그레이션하려는 경우 다음 절차를 사용하십시오.

실행 중인 서버의 스냅샷을 저장하여 새 이미지를 생성하고 이미지를 다운로드합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack server image create instance_name --name image_name
(overcloud) $ openstack image save image_name --file exported_vm.qcow2
```

오버클라우드로 내보낸 이미지를 업로드하고 새 인스턴스를 시작합니다.

```
(overcloud) $ openstack image create imported_image --file exported_vm.qcow2 --disk-format qcow2
--container-format bare
(overcloud) $ openstack server create  imported_instance --key-name default --flavor m1.demo --
image imported_image --nic net-id=net_id
```

중요

각 VM 디스크를 기존 OpenStack 환경에서 새 Red Hat OpenStack Platform으로 복사해야 합니다. QCOW를 사용한 스냅샷은 원래 계층 시스템이 손실됩니다.

9.11. 오버클라우드 삭제 방지

Heat에는 **/etc/heat/policy.json**을 생성하고 사용자 지정 규칙을 추가하여 재정의할 수 있는 기본 정책 세트가 코드로 포함되어 있습니다. 모든 *사용자*에 대해 오버클라우드 삭제 권한을 해제하려면 다음 정책을 추가합니다.

```
{"stacks:delete": "rule:deny_everybody"}
```

이렇게 하면 오버클라우드가 **heat** 클라이언트와 함께 삭제되지 않습니다. 오버클라우드를 삭제하려면 사용자 지정 정책을 삭제하고 **/etc/heat/policy.json**을 저장합니다.

9.12. 오버클라우드 제거

원하는 경우 전체 오버클라우드를 제거할 수 있습니다.

기존 오버클라우드를 삭제합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud delete overcloud
```

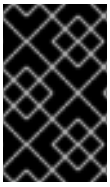
오버클라우드 삭제를 확인합니다.

```
(undercloud) $ openstack stack list
```

삭제에는 몇 분 정도 시간이 걸립니다.

삭제가 완료되면 배포 시나리오의 표준 단계에 따라 오버클라우드를 다시 생성합니다.

10장. ANSIBLE로 오버클라우드 구성



중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 Red Hat에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

Ansible을 기본 방법으로 사용하여 오버클라우드 구성을 적용할 수 있습니다. 이 장에서는 오버클라우드에서 이 기능을 활성화하는 방법을 설명합니다.

director가 Ansible 플레이북을 자동으로 생성하지만 Ansible 구문을 숙지하는 것이 좋습니다. Ansible 사용 방법에 대한 자세한 내용은 https://docs.ansible.com/을 참조하십시오.

참고

Ansible은 OpenStack Platform director 역할과 다른 역할의 개념을 사용합니다.

참고

이 설정 방법은 노드에 Ceph Storage 클러스터 배포를 지원하지 않습니다.

10.1. ANSIBLE 기반 오버클라우드 구성(CONFIG-DOWNLOAD)

config-download 기능:

- Heat 대신 Ansible을 사용하여 오버클라우드 구성 애플리케이션을 활성화합니다.
- 오버클라우드 노드의 Heat 및 Heat 에이전트(**os-collect-config**) 간 구성 배포 데이터의 통신 및 전송

heat는 **config-download** 를 활성화하거나 사용하지 않고 표준 기능을 유지합니다.

- director는 환경 파일과 매개 변수를 Heat에 전달합니다.
- director는 Heat를 사용하여 스택 및 모든 하위 리소스를 만듭니다.
- Heat는 베어 메탈 노드 및 네트워크 생성을 포함하여 모든 OpenStack 서비스 리소스를 생성합니다.

Heat는 **SoftwareDeployment** 리소스에서 모든 배포 데이터를 생성하여 오버클라우드 설치 및 구성을 수행하지만 설정을 적용하지는 않습니다. 대신 Heat는 API를 통해 데이터만 제공합니다. 스택이 생성되면 Mistral 워크플로우는 배포 데이터에 대한 Heat API를 쿼리하고 Ansible 인벤토리 파일 및 생성된 플레이북 세트로 **ansible-playbook** 을 실행하여 구성을 적용합니다.

10.2. 오버클라우드 설정 방법을 CONFIG-DOWNLOAD로 전환

다음 절차에서는 OpenStack Orchestration(heat)에서 Ansible 기반 **config-download** 메서드로 오버클라우드 구성 방법을 전환합니다. 이 경우 언더클라우드는 Ansible 제어 노드 i.e. **ansible-playbook** 을 실행하는 노드 역할을 합니다. 제어 노드 와 언더클라우드라는 용어는 언더클라우드 설치를 수행한 동일한 노드를 나타냅니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 오버클라우드 배포 명령을 실행하고 **--config-download** 옵션과 환경 파일을 포함하여 heat 기반 설정을 비활성화합니다.

```
$ openstack overcloud deploy --templates \
   --config-download \
   -e /usr/share/openstack-tripleo-heat-templates/environments/config-download-
environment.yaml \
   --overcloud-ssh-user heat-admin \
   --overcloud-ssh-key ~/.ssh/id_rsa \
   [OTHER OPTIONS]
```

다음 옵션을 사용합니다.

- **--config-download** 를 사용하면 추가 Mistral 워크플로우를 활성화하여 Heat 대신**ansible-playbook** 으로 구성을 적용합니다.
- **-e /usr/share/openstack-tripleo-heat-templates/environments/config-download-environment.yaml** 은 Heat 소프트웨어 배포 구성 리소스를 Ansible 기반 동등한 항목에 매핑하는 필수 환경 파일입니다. 이렇게 하면 설정을 적용하지 않고 Heat API를 통해 구성 데이터를 제공합니다.
- **--overcloud-ssh-user** 및 **--overcloud-ssh-key** 는 각 오버클라우드 노드에 SSH를 사용하고, 초기 **tripleo-admin** 사용자를 생성한 다음 SSH 키를**/home/tripleo-admin/.ssh/authorized_keys** 에 삽입하는 데 사용됩니다. SSH 키를 삽입하기 위해 사용자는 **--overcloud-ssh-user** (defaults to**heat-admin**) 및 **--overcloud-ssh-key** (defaults to **~/.ssh/id_rsa**)를 사용하여 초기 SSH 연결에 대한 자격 증명을 지정합니다.**--overcloud-ssh-key** 로 지정된 개인 키로 공개를 제한하기 위해 director는 Heat 또는 Mistral과 같은 API 서비스에 이 키를 전달하지 않으며 director의 **openstack overcloud deploy** 명령만 이 키를 사용하여 **tripleo-admin** 사용자의 액세스를 활성화합니다.

이 명령을 실행하는 경우 오버클라우드와 관련된 기타 파일도 포함해야 합니다. 예를 들면 다음과 같습니다.

- **-e**를 사용하여 사용자 정의 구성 환경 파일
- **--roles-file**을 사용하는 사용자 지정 역할(**roles_data**) 파일
- **--networks-file**을 사용하는 구성 가능 네트워크(**network_data**) 파일

3. 오버클라우드 배포 명령은 표준 스택 작업을 수행합니다. 그러나 오버클라우드 스택이 설정 단계에 도달하면 스택이 오버클라우드 구성을 위해 **config-download** 메서드로 전환합니다.

```
2018-05-08 02:48:38Z [overcloud-AllNodesDeploySteps-xzihzsekhwo6]:
UPDATE_COMPLETE  Stack UPDATE completed successfully
2018-05-08 02:48:39Z [AllNodesDeploySteps]: UPDATE_COMPLETE  state changed
2018-05-08 02:48:45Z [overcloud]: UPDATE_COMPLETE  Stack UPDATE completed
successfully

 Stack overcloud UPDATE_COMPLETE

Deploying overcloud configuration
```

오버클라우드 구성이 완료될 때까지 기다립니다.

4. 오버클라우드의 Ansible 설정이 완료되면 director에서 성공적이고 실패한 작업 및 오버클라우드에 대한 액세스 URL에 대한 보고서를 제공합니다.

```
PLAY RECAP *************************************************************
192.0.2.101        : ok=173  changed=42   unreachable=0    failed=0
192.0.2.102        : ok=133  changed=42   unreachable=0    failed=0
localhost          : ok=2    changed=0    unreachable=0    failed=0


Ansible passed.
Overcloud configuration completed.
Started Mistral Workflow tripleo.deployment.v1.get_horizon_url. Execution ID: 0e4ca4f6-
9d14-418a-9c46-27692649b584
Overcloud Endpoint: http://10.0.0.1:5000/
Overcloud Horizon Dashboard URL: http://10.0.0.1:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

사전 프로비저닝된 노드를 사용하는 경우 **config-download** 를 사용하여 성공적으로 배포할 수 있도록 추가 단계를 수행해야 합니다.

10.3. 사전 프로비저닝된 노드를 사용하여 CONFIG-DOWNLOAD 활성화

사전 프로비저닝된 노드와 함께 **config-download** 를 사용하는 경우**ansible-playbook** 이 확인 가능한 호스트에 도달할 수 있도록 Heat 기반 호스트 이름을 실제 호스트 이름에 매핑해야 합니다. **HostnameMap**을 사용하여 이러한 값을 매핑할 수 있습니다.

절차

1. 환경 파일(예: **hostname-map.yaml**)을 생성하고 **HostnameMap** 매개변수와 호스트 이름 매핑을 포함합니다. 다음 구문을 사용합니다.

```
parameter_defaults:
  HostnameMap:
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
```

[HEAT HOSTNAME] 은 일반적으로 **[STACK NAME]-[ROLE]-[INDEX]** 규칙을 따릅니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  HostnameMap:
    overcloud-controller-0: controller-00-rack01
    overcloud-controller-1: controller-01-rack02
    overcloud-controller-2: controller-02-rack03
    overcloud-novacompute-0: compute-00-rack01
    overcloud-novacompute-1: compute-01-rack01
    overcloud-novacompute-2: compute-02-rack01
```

2. **hostname-map.yaml** 의 콘텐츠를 저장합니다.

3. **config-download** 배포를 실행하는 경우 **-e** 옵션을 사용하여 환경 파일을 포함합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  --config-download \
  -e /usr/share/openstack-tripleo-heat-templates/environments/config-download-
environment.yaml \
  -e /home/stack/templates/hostname-map.yaml \
  --overcloud-ssh-user heat-admin \
  --overcloud-ssh-key ~/.ssh/id_rsa \
  [OTHER OPTIONS]
```

10.4. CONFIG-DOWNLOAD 작업 디렉터리에 대한 액세스 활성화

mistral은 config-download 기능을 위해 Ansible 플레이북을 실행합니다. mistral은 플레이북, 구성 파일 및 로그를 작업 디렉터리에 저장합니다. **/var/lib/mistral/** 에서 이러한 작업 디렉터리를 찾을 수 있으며 Mistral 워크플로우 실행 UUID를 사용하여 이름이 지정됩니다.

이러한 작업 디렉터리에 액세스하기 전에 **stack** 사용자에게 적절한 권한을 설정해야 합니다.

절차

1. **mistral** 그룹은 **/var/lib/mistral** 아래의 모든 파일을 읽을 수 있습니다. 언더클라우드의 대화형 **stack** 사용자에게 해당 파일에 대한 읽기 전용 액세스 권한을 부여합니다.

```
$ sudo usermod -a -G mistral stack
```

2. 다음 명령을 사용하여 **stack** 사용자의 권한을 새로 고칩니다.

```
[stack@director ~]$ exec su -l stack
```

명령을 실행하면 다시 로그인하라는 메시지가 표시됩니다. **stack** 사용자의 암호를 입력합니다.

3. **/var/lib/mistral** 디렉터리에 대한 읽기 액세스를 테스트합니다.

```
$ ls /var/lib/mistral/
```

10.5. CONFIG-DOWNLOAD 로그 및 작업 디렉터리 확인

config-download 프로세스 중에 Ansible은 **/var/lib/mistral/<execution uuid>/ansible.log** 에 로그 파일을 생성합니다. **<execution uuid>** 는 **ansible-playbook** 을 실행한 Mistral 실행에 해당하는 UUID입니다.

절차

1. **openstack workflow execution list** 명령을 사용하여 모든 실행을 나열하고 **config-download** 를 실행하는 선택한 Mistral 실행의 워크플로우 ID를 찾습니다.

```
$ openstack workflow execution list
$ less /var/lib/mistral/<execution uuid>/ansible.log
```

<execution uuid> 는 **ansible-playbook** 을 실행한 Mistral 실행의 UUID입니다.

2. 또는 **/var/lib/mistral** 에서 가장 최근에 수정된 디렉토리를 찾아 가장 최근 배포의 로그를 빠르게 찾습니다.

```
$ less /var/lib/mistral/$(ls -t /var/lib/mistral | head -1)/ansible.log
```

10.6. CONFIG-DOWNLOAD를 수동으로 실행

/var/lib/mistral/ 의 각 작업 디렉터리에는**ansible-playbook** 과 직접 상호 작용하는 데 필요한 플레이북과 스크립트가 포함되어 있습니다. 다음 절차에서는 이러한 파일과 상호작용하는 방법을 설명합니다.

절차

1. 선택한 Ansible 플레이북의 디렉터리로 변경합니다.

```
$ cd /var/lib/mistral/<execution uuid>/
```

<execution uuid> 는 **ansible-playbook** 을 실행한 Mistral 실행의 UUID입니다.

2. mistral 작업 디렉터리에서 **ansible-playbook-command.sh** 를 실행하여 배포를 재현합니다.

```
$ ./ansible-playbook-command.sh
```

3. Ansible 인수를 이 스크립트에 추가로 전달할 수 있으며, 이러한 인수는 변경되지 않은 상태로 **ansible-playbook** 명령에 전달됩니다. 이를 통해 확인 모드(**--check**), 호스트 제한(**--limit**), 변수 덮어쓰기(**-e**) 등의 Ansible 기능을 더욱 효과적으로 활용할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ ./ansible-playbook-command.sh --limit Controller
```

4. 작업 디렉터리에는 **deploy_steps_playbook.yaml**이라는 플레이북이 포함되어 있으며 이는 오버클라우드를 실행합니다. 이 플레이북을 보려면 다음을 수행합니다.

```
$ less deploy_steps_playbook.yaml
```

플레이북은 작업 디렉터리에 포함된 다양한 작업 파일을 사용합니다. 일부 작업 파일은 모든 OpenStack Platform 역할에 공통되며 일부 파일은 특정 OpenStack Platform 역할과 서버에 한정되어 있습니다.

5. 또한 작업 디렉터리에는 사용자 오버클라우드의 **roles_data** 파일에 정의된 각 역할에 해당하는 하위 디렉터리가 포함되어 있습니다. 예를 들면 다음과 같습니다.

```
$ ls Controller/
```

각 OpenStack Platform 역할 디렉터리에는 해당 역할 유형의 개별 서버에 대한 하위 디렉터리가 포함되어 있습니다. 디렉터리는 구성 가능 역할 호스트 이름 포맷을 사용합니다. 예를 들면 다음과 같습니다.

```
$ ls Controller/overcloud-controller-0
```

6. Ansible 작업이 태그됩니다. 전체 태그 목록을 확인하려면 **ansible-playbook**에 대해 CLI 인수**--list-tags**를 사용합니다.

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
```

-

다음으로 **ansible-playbook-command.sh** 스크립트에서 **--tags, --skip-tags** 또는 **--start-at-task**를 사용하여 태그된 구성을 적용합니다. 예를 들면 다음과 같습니다.

```
$ ./ansible-playbook-command.sh --tags overcloud
```

주의

--tags, --skip-tags 또는 **--start-at-task**와 같은 ansible-playbook CLI 인수를 사용할 때 순서가 잘못된 배포를 실행하거나 적용하지 마십시오. CLI 인수는 이전에 실패한 작업을 재실행하거나 초기 배포를 반복하는 편리한 방법입니다. 하지만 일관된 배포를 위해 **deploy_steps_playbook.yaml**의 모든 작업을 순서대로 실행해야 합니다.

10.7. CONFIG-DOWNLOAD 비활성화

표준 Heat 기반 구성 방법으로 다시 전환하려면 다음에 **openstack overcloud deploy** 를 실행할 때 관련 옵션 및 환경 파일을 제거하십시오.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 오버클라우드 배포 명령을 실행하되 **--config-download** 옵션 또는 'config-download-environment.yaml 환경 파일은 포함하지 마십시오.

```
$ openstack overcloud deploy --templates \
    [OTHER OPTIONS]
```

이 명령을 실행하는 경우 오버클라우드와 관련된 기타 파일도 포함해야 합니다. 예를 들면 다음과 같습니다.

- **-e**를 사용하여 사용자 정의 구성 환경 파일
- **--roles-file**을 사용하는 사용자 지정 역할(**roles_data**) 파일
- **--networks-file**을 사용하는 구성 가능 네트워크(**network_data**) 파일

3. 오버클라우드 배포 명령은 Heat의 구성을 포함하여 표준 스택 작업을 수행합니다.

10.8. 다음 단계

이제 정상적인 오버클라우드 작업을 계속할 수 있습니다.

11장. 가상화된 컨트롤 플레인 생성

가상화된 컨트롤 플레인은 베어 메탈이 아닌 VM(가상 머신)에 있는 컨트롤 플레인입니다. 가상화된 컨트롤 플레인은 컨트롤 플레인에 필요한 베어 메탈 머신의 수를 줄입니다.

이 장에서는 RHOSP(Red Hat OpenStack Platform) 및 Red Hat Virtualization을 사용하여 RHOSP 컨트롤 플레인을 가상화하는 방법을 설명합니다.

11.1. 가상화된 컨트롤 플레인 아키텍처

RHOSP(Red Hat OpenStack Platform) director를 사용하여 Red Hat Virtualization 클러스터에 배포된 컨트롤러 노드를 사용하여 오버클라우드를 프로비저닝합니다. 그런 다음 가상화된 컨트롤러를 가상화된 컨트롤 플레인 노드로 배포할 수 있습니다.

참고

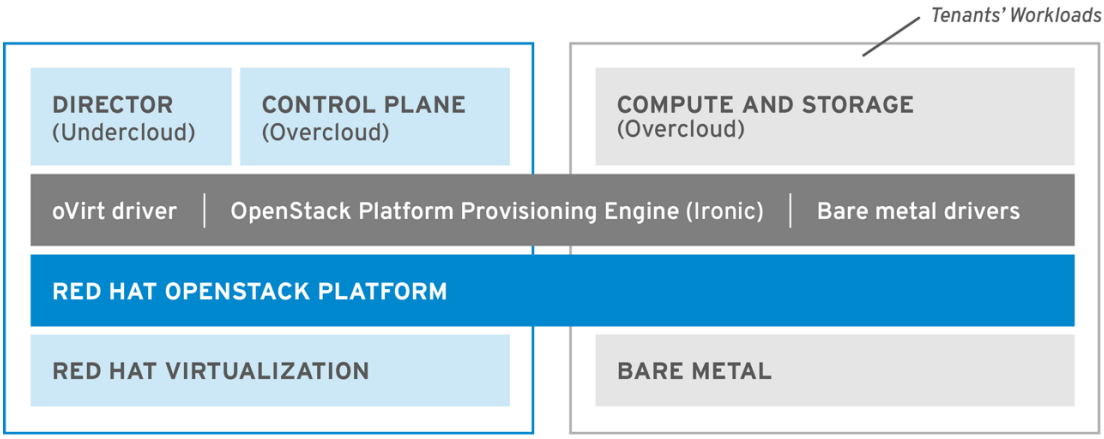
가상화된 컨트롤러 노드는 Red Hat Virtualization에서만 지원됩니다.

다음 아키텍처 다이어그램에서는 가상화된 컨트롤 플레인 배포 방법을 설명합니다. Red Hat Virtualization의 VM에서 실행 중인 컨트롤러 노드로 오버클라우드를 배포합니다. 베어 메탈에서 컴퓨팅 및 스토리지 노드를 실행합니다.

참고

Red Hat Virtualization에서 OpenStack 가상화된 언더클라우드를 실행합니다.

가상화된 컨트롤 플레인 아키텍처



OpenStack Bare Metal Provisioning(ironic) 서비스에는 Red Hat Virtualization VM, staging-ovirt의 드라이버가 포함되어 있습니다. Red Hat Virtualization 환경에서 가상 노드를 관리하는 데 이 드라이버를 사용할 수 있습니다. 이 드라이버를 사용하여 오버클라우드 컨트롤러를 Red Hat Virtualization 환경 내의 가상 머신으로 배포할 수 도 있습니다.

11.2. RHOSP 오버클라우드 컨트롤 플레인 가상화의 이점과 제한 사항

RHOSP 오버클라우드 컨트롤 플레인을 가상화하는 이점은 여러 가지가 있지만 모든 구성에서 선택할 수 있는 것은 아닙니다.

이점

오버클라우드 컨트롤 플레인을 가상화하면 가동 중단 시간을 방지하고 성능을 향상시키는 여러 이점이 있습니다.

- 핫 애드 및 핫 삭제를 사용하여 필요에 따라 CPU 및 메모리를 확장할 수 있도록 가상화된 컨트롤러에 리소스를 동적으로 할당할 수 있습니다. 그러면 가동 중단 시간을 방지하고 플랫폼이 증가함에 따라 쉽게 용량을 늘릴 수 있습니다.
- 동일한 Red Hat Virtualization 클러스터에서 추가 인프라 VM을 배포할 수 있습니다. 이는 데이터 센터의 서버 공간 사용을 최소화하고 물리적 노드의 효율성을 극대화합니다.
- 구성 가능 역할을 사용하여 추가 복합 RHOSP 컨트롤 플레인을 정의할 수 있습니다. 이를 통해 컨트롤 플레인의 특정 구성 요소에 리소스를 할당할 수 있습니다.
- VM 실시간 마이그레이션 기능을 활용하여 서비스 중단 없이 시스템을 유지보수할 수 있습니다.
- Red Hat Virtualization에서 지원하는 타사 또는 사용자 지정 툴을 통합할 수 있습니다.

제한 사항

가상화된 컨트롤 플레인에서는 사용할 수 있는 구성 유형이 제한됩니다.

- 가상화된 Ceph Storage 노드와 컴퓨팅 노드는 지원되지 않습니다.
- 파이버 채널을 사용하는 백엔드에는 블록 스토리지(cinder) 이미지-볼륨(image-to-volume)이 지원되지 않습니다. Red Hat Virtualization에서 NPIV(N_Port ID Virtualization)를 지원하지 않습니다. 따라서 스토리지 백엔드의 LUN을 cinder-volume이 기본적으로 실행되는 컨트롤러에 매핑해야 하는 블록 스토리지(cinder) 드라이버가 작동하지 않습니다. 가상화된 컨트롤러에 cinder-volume을 포함하지 않고 전용 역할을 생성하는 것이 좋습니다. 자세한 내용은Composable Services and Custom Roles를 참조하십시오.

11.3. RED HAT VIRTUALIZATION 드라이버를 사용하여 가상화된 컨트롤러 프로비저닝

이 절에서는 RHOSP 및 Red Hat Virtualization을 사용하여 오버클라우드의 가상화된 RHOSP 컨트롤 플레인을 프로비저닝하는 방법을 자세히 설명합니다.

사전 요구 사항

- Intel 64 또는 AMD64 CPU 확장 기능을 지원하는 64비트 x86 프로세서가 있어야 합니다.
- 다음 소프트웨어가 이미 설치되어 설정되어 있어야 합니다.
  - Red Hat Virtualization 자세한 내용은 Red Hat Virtualization Documentation Suite를 참조하십시오.
  - RHOSP(Red Hat OpenStack Platform). 자세한 내용은 Director Installation and Usage를 참조하십시오.
- 가상화된 컨트롤러 노드가 미리 준비되어 있어야 합니다. 이 요구 사항은 베어 메탈 컨트롤러 노드에도 마찬가지입니다. 자세한 내용은 컨트롤러 노드 요구 사항을 참조하십시오.
- 베어 메탈 노드가 오버클라우드 컴퓨팅 노드로 사용 중이어야 하며 스토리지 노드가 미리 준비되어 있어야 합니다. 하드웨어 사양은 컴퓨팅 노드 요구 사항 및 Ceph Storage 노드 요구 사항을 참조하십시오. POWER(ppc64le) 하드웨어에 오버클라우드 컴퓨팅 노드를 배포하려면 Red Hat OpenStack Platform for POWER를 참조하십시오.

- 논리 네트워크가 생성되었으며, 클러스터 또는 호스트 네트워크가 여러 네트워크에서 네트워크 분리를 사용할 준비가 되어 있어야 합니다. 자세한 내용은 Logical Networks를 참조하십시오.
- 각 노드의 내부 BIOS 시계를 UTC로 설정해야 합니다. 이렇게 하면 시간대 오프셋을 적용하기 전에 hwclock이 BIOS 시계를 동기화할 때 미래 날짜의 파일 타임스탬프 관련 문제를 방지할 수 있습니다.

작은 정보

성능 장애를 방지하려면 구성 가능 역할을 사용하고 베어 메탈 컨트롤러 노드에 데이터 플레인 서비스를 유지합니다.

절차

1. **undercloud.conf** 설정 파일의 **enabled_hardware_types**에 드라이버를 추가하여 director 언더클라우드에서 **staging-ovirt** 드라이버를 활성화합니다.

```
enabled_hardware_types = ipmi,redfish,ilo,idrac,staging-ovirt
```

2. 언더클라우드에 **staging-ovirt** 드라이버가 있는지 확인합니다.

```
(undercloud) [stack@undercloud ~]$ openstack baremetal driver list
```

언더클라우드가 제대로 설정되어 있으면 명령에서 다음 결과가 반환됩니다.

```
+---------------------+----------------------+
| Supported driver(s) | Active host(s)       |
+---------------------+----------------------+
| idrac               | localhost.localdomain |
| ilo                 | localhost.localdomain |
| ipmi                | localhost.localdomain |
| pxe_drac            | localhost.localdomain |
| pxe_ilo             | localhost.localdomain |
| pxe_ipmitool        | localhost.localdomain |
| redfish             | localhost.localdomain |
| staging-ovirt       | localhost.localdomain |
```

3. **python-ovirt-engine-sdk4.x86_64** 패키지를 설치합니다.

```
$ sudo yum install python-ovirt-engine-sdk4
```

4. 오버클라우드 노드 정의 템플릿(예: **nodes.json**)을 업데이트하여 Red Hat Virtualization에 호스팅된 VM을 director에 등록합니다. 자세한 내용은 오버클라우드용 노드 등록을 참조하십시오. 다음 키:값 쌍을 사용하여 오버클라우드와 함께 배포하려는 VM의 속성을 정의합니다.

표 11.1. 오버클라우드에 대한 VM 설정

키	이 값으로 설정
pm_type	oVirt/RHV VM용 OpenStack Bare Metal Provisioning(ironic) 서비스 드라이버인 **staging-ovirt**

키	이 값으로 설정
pm_user	Red Hat Virtualization Manager 사용자 이름.
pm_password	Red Hat Virtualization Manager 암호.
pm_addr	Red Hat Virtualization Manager 서버의 호스트 이름 또는 IP.
pm_vm_name	Red Hat Virtualization Manager에서 컨트롤러가 생성된 가상 머신의 이름.

예를 들면 다음과 같습니다.

```
{
    "nodes": [
        {
            "name":"osp13-controller-0",
            "pm_type":"staging-ovirt",
            "mac":[
                "00:1a:4a:16:01:56"
            ],
            "cpu":"2",
            "memory":"4096",
            "disk":"40",
            "arch":"x86_64",
            "pm_user":"admin@internal",
            "pm_password":"password",
            "pm_addr":"rhvm.example.com",
            "pm_vm_name":"{vernum}-controller-0",
            "capabilities": "profile:control,boot_option:local"
        },
}
```

각 Red Hat Virtualization 호스트에서 하나의 컨트롤러 설정

5. Red Hat Virtualization에서 "soft negative affinity"로 선호도 그룹을 설정하여 컨트롤러 VM에 대해 고가용성이 구현되었는지 확인합니다. 자세한 내용은 Affinity Groups를 참조하십시오.

6. Red Hat Virtualization Manager 인터페이스를 열고 이를 사용하여 각 VLAN을 컨트롤러 VM의 개별 논리 vNIC에 매핑합니다. 자세한 내용은 Logical Networks를 참조하십시오.

7. director와 컨트롤러 VM의 vNIC에서 **no_filter**를 설정하고 VM을 다시 시작하여 컨트롤러 VM에 연결된 네트워크에서 MAC 스푸핑 필터를 비활성화합니다. 자세한 내용은 Virtual Network Interface Cards를 참조하십시오.

8. 오버클라우드를 배포하여 새 가상 컨트롤러 노드를 환경에 추가합니다.

```
(undercloud) [stack@undercloud ~]$ openstack overcloud deploy --templates
```

12장. 오버클라우드 노드 확장

오버클라우드 생성 후 노드를 추가하거나 삭제하려면 오버클라우드를 업데이트해야 합니다.

주의

오버클라우드에서 노드를 삭제하는 데 openstack server delete를 사용하지 마십시오. 이 섹션에서 설명하는 절차를 확인하여 노드를 적절하게 삭제하고 교체합니다.

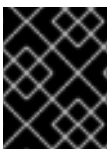
참고

오버클라우드 노드를 확장하거나 제거하기 전에 베어 메탈 노드가 유지보수 모드에 없는지 확인합니다.

아래 표를 사용하여 각 노드 유형의 확장 지원 여부를 확인합니다.

표 12.1. 각 노드 유형의 확장 지원

노드 유형	확장 가능 여부	축소 가능 여부	비고
컨트롤러	N	N	13장. *컨트롤러 노드 교체*의 절차를 사용하여 컨트롤러 노드를 교체할 수 있습니다.
Compute	Y	Y	
Ceph Storage 노드	Y	N	초기 오버클라우드 생성 시 적어도 하나의 Ceph Storage 노드가 있어야 합니다.
Object Storage 노드	Y	Y	



중요

오버클라우드를 확장하려면 적어도 10GB의 여유 공간이 있어야 합니다. 이 공간은 노드 프로비저닝 프로세스 중에 이미지 변환 및 캐싱에 사용됩니다.

12.1. 오버클라우드에 노드 추가

director 노드 풀에 노드를 추가하려면 다음 단계를 완료합니다.

절차

1. **등록할 새 노드 세부 정보가 포함된 새 JSON 파일(newnodes.json)을 생성합니다.**

```
{
  "nodes":[
    {
        "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":"192.168.24.207"
    },
    {
        "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":"192.168.24.208"
    }
  ]
}
```

2. 다음 명령을 실행하여 새 노드를 등록합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud node import newnodes.json
```

3. 새 노드를 등록한 후 다음 명령을 실행하여 노드를 나열하고 새 노드 UUID를 확인합니다.

```
(undercloud) $ openstack baremetal node list
```

4. 다음 명령을 실행하여 각 새 노드에 인트로스펙션 프로세스를 시작합니다.

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

이 프로세스에서 노드의 하드웨어 속성을 감지하여 벤치마킹합니다.

5. 노드의 이미지 속성을 설정합니다.

```
(undercloud) $ openstack overcloud node configure [NODE UUID]
```

12.2. 역할의 노드 수 추가

컴퓨팅 노드와 같은 특정 역할의 오버클라우드 노드를 확장하려면 다음 단계를 완료합니다.

절차

1. 원하는 역할로 각각의 새 노드를 태그합니다. 예를 들면 컴퓨팅 역할로 노드를 태그하고 다음 명령을 실행합니다.

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:compute,boot_option:local' [NODE UUID]
```

2. 오버클라우드를 확장하려면 노드 수를 포함하는 환경 파일을 편집하고 오버클라우드를 다시 배포해야 합니다. 예를 들어 오버클라우드를 컴퓨팅 노드 5개로 확장하려면 **ComputeCount** 매개변수를 편집합니다.

```
parameter_defaults:
  ...
  ComputeCount: 5
  ...
```

3. 업데이트된 파일로 배포 명령을 재실행합니다. 이 예제에서는 **node-info.yaml**이라고 합니다.

```
(undercloud) $ openstack overcloud deploy --templates -e /home/stack/templates/node-
info.yaml [OTHER_OPTIONS]
```

초기 오버클라우드 생성 시의 모든 환경 파일과 옵션을 포함해야 합니다. 여기에는 컴퓨팅 이외의 노드에 대한 동일한 확장 매개변수가 포함됩니다.

4. 배포 작업이 완료될 때까지 기다립니다.

12.3. 컴퓨팅 노드 제거 또는 교체

경우에 따라 오버클라우드에서 컴퓨팅 노드를 삭제해야 합니다. 예를 들면 문제가 있는 컴퓨팅 노드를 교체해야 할 수 있습니다. 컴퓨팅 노드를 삭제하면 확장 작업 중에 인덱스가 재사용되지 않도록 노드의 인덱스가 기본적으로 거부 목록에 추가됩니다.

오버클라우드 배포에서 노드를 삭제한 후 삭제된 컴퓨팅 노드를 교체할 수 있습니다.

사전 요구 사항

- 제거하려는 노드에서 Compute 서비스가 비활성화되어 노드가 새 인스턴스가 예약되지 않습니다. Compute 서비스가 비활성화되었는지 확인하려면 다음 명령을 사용하십시오.

```
(overcloud)$ openstack compute service list
```

Compute 서비스를 비활성화하지 않으면 비활성화합니다.

```
(overcloud)$ openstack compute service set <hostname> nova-compute --disable
```

작은 정보

--disable-reason 옵션을 사용하여 서비스가 비활성화되는 이유에 대한 간단한 설명을 추가합니다. 이 기능은 Compute 서비스를 재배포하려는 경우에 유용합니다.

- 컴퓨팅 노드의 워크로드가 다른 컴퓨팅 노드로 마이그레이션되었습니다. 자세한 내용은 Migrating virtual machine instances between Compute nodes를 참조하십시오.
- Instance HA가 활성화된 경우 다음 옵션 중 하나를 선택합니다.
  - 컴퓨팅 노드에 액세스할 수 있으면 **root** 사용자로 컴퓨팅 노드에 로그인하고**shutdown -h now** 명령을 사용하여 완전히 종료합니다.
  - 컴퓨팅 노드에 액세스할 수 없는 경우 컨트롤러 노드에 **root** 사용자로 로그인하여 컴퓨팅 노드의 STONITH 장치를 비활성화한 다음 베어 메탈 노드를 종료합니다.

```
[root@controller-0 ~]# pcs stonith disable <stonith_resource_name>
[stack@undercloud ~]$ source stackrc
[stack@undercloud ~]$ openstack baremetal node power off <UUID>
```

절차

1. **source** 명령으로 언더클라우드 설정을 로드합니다.

```
(overcloud)$ source ~/stackrc
```

2. 오버클라우드 스택의 UUID를 확인합니다.

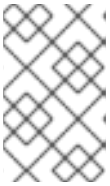
```
(undercloud)$ openstack stack list
```

3. 삭제하려는 컴퓨팅 노드의 UUID 또는 호스트 이름을 확인합니다.

```
(undercloud)$ openstack server list
```

4. 선택 사항: **--update-plan-only** 옵션과 함께 **overcloud deploy** 명령을 실행하여 템플릿에서 최신 구성으로 플랜을 업데이트합니다. 이렇게 하면 컴퓨팅 노드를 삭제하기 전에 오버클라우드 구성이 최신 상태로 유지됩니다.

```
$ openstack overcloud deploy --update-plan-only \
  --templates  \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/rhel-registration/environment-rhel-registration.yaml \
  [-e |...]
```



참고

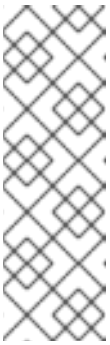
오버클라우드 노드 거부 목록을 업데이트한 경우 이 단계가 필요합니다. 오버클라우드 노드를 거부 목록에 추가하는 방법에 대한 자세한 내용은 노드 블랙리스트 지정을 참조하십시오.

5. 스택에서 컴퓨팅 노드를 삭제합니다.

```
$ openstack overcloud node delete --stack <overcloud> \
 <node_1> ... [node_n]
```

- **<overcloud>**를 오버클라우드 스택의 이름 또는 UUID로 바꿉니다.

- & lt;node_1 >을 바꾸고 선택적으로[node_n] 까지 모든 노드를 삭제하려는 컴퓨팅 노드의 Compute 서비스 호스트 이름 또는 UUID로 바꿉니다. UUID와 호스트 이름을 혼합하여 사용하지 마십시오. UUID만 사용하거나 호스트 이름만 사용하십시오.



참고

노드의 전원이 이미 꺼진 경우 이 명령은 **WARNING** 메시지를 반환합니다.

```
Ansible failed, check log at /var/lib/mistral/overcloud/ansible.log
WARNING: Scale-down configuration error. Manual cleanup of some
actions may be necessary. Continuing with node removal.
```

이 메시지는 무시해도 됩니다.

6. 컴퓨팅 노드가 삭제될 때까지 기다립니다.

7. 노드 삭제가 완료되면 오버클라우드 스택의 상태를 확인합니다.

```
(undercloud)$ openstack stack list
```

표 12.2. 결과

상태	설명
UPDATE_COMPLETE	삭제 작업이 성공적으로 완료되었습니다.
UPDATE_FAILED	삭제 작업이 실패했습니다. 삭제에 실패한 일반적인 이유는 삭제 작업을 삭제하려는 노드의 IPMI 인터페이스에 연결할 수 없기 때문입니다. 삭제 작업이 실패하면 컴퓨팅 노드를 수동으로 삭제해야 합니다. 자세한 내용은 컴퓨팅 노드 수동 제거를 참조하십시오.

8. 인스턴스 HA가 활성화된 경우 다음 작업을 수행합니다.

   a. 노드의 Pacemaker 리소스를 정리합니다.

```
$ sudo pcs resource delete <scaled_down_node>
$ sudo cibadmin -o nodes --delete --xml-text '<node id="<scaled_down_node>"/>'
$ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete --xml-text '<node_state id="<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete-all --xml-text '<node id="<scaled_down_node>"/>' --
force
```

   b. 노드의 STONITH 장치를 삭제합니다.

```
$ sudo pcs stonith delete <device-name>
```

9. 오버클라우드에서 삭제된 Compute 노드를 교체하지 않는 경우 노드 수가 포함된 환경 파일의 **ComputeCount** 매개변수를 줄입니다. 일반적으로 이 파일의 이름은**node-info.yaml**로 지정됩니다. 예를 들어 한 노드를 삭제한 경우 노드 수를 4개의 노드에서 3개의 노드로 줄입니다.

```
parameter_defaults:
  ...
  ComputeCount: 3
```

노드 개수를 줄이면 **openstack overcloud deploy**를 실행할 때 director에서 새 노드를 프로비저닝하지 않습니다.

오버클라우드 배포에서 삭제된 컴퓨팅 노드를 교체하는 경우 삭제된 컴퓨팅 노드 배치를참조하십시오.

12.3.1. 컴퓨팅 노드 수동 제거

연결할 수 없는 노드로 인해 **openstack overcloud node delete** 명령이 실패한 경우 오버클라우드에서 컴퓨팅 노드 제거를 수동으로 완료해야 합니다.

사전 요구 사항

- Compute 노드 프로시저 제거 또는 교체를 수행하면**UPDATE_FAILED** 상태가 반환되었습니다.

절차

1. 오버클라우드 스택의 UUID를 확인합니다.

```
(undercloud)$ openstack stack list
```

2. 수동으로 삭제하려는 노드의 UUID를 확인합니다.

```
(undercloud)$ openstack baremetal node list
```

3. 삭제할 노드를 유지보수 모드로 이동합니다.

```
(undercloud)$ openstack baremetal node maintenance set <node_uuid>
```

4. Compute 서비스가 베어 메탈 서비스와 상태를 동기화할 때까지 기다립니다. 이 작업은 최대 4분이 걸릴 수 있습니다.

5. source 명령으로 오버클라우드 설정을 로드합니다.

```
(undercloud)$ source ~/overcloudrc
```

6. 삭제한 노드의 네트워크 에이전트를 삭제합니다.

```
(overcloud)$ for AGENT in $(openstack network agent list --host <scaled_down_node> -c ID
-f value) ; do openstack network agent delete $AGENT ; done
```

<scaled_down_node>를 제거할 노드 이름으로 바꿉니다.

7. 노드에서 새 인스턴스가 예약되지 않도록 오버클라우드의 삭제된 노드에서 Compute 서비스가 비활성화되었는지 확인합니다.

```
(overcloud)$ openstack compute service list
```

Compute 서비스를 비활성화하지 않으면 비활성화합니다.

```
(overcloud)$ openstack compute service set <hostname> nova-compute --disable
```

작은 정보

--disable-reason 옵션을 사용하여 서비스가 비활성화되는 이유에 대한 간단한 설명을 추가합니다. 이 기능은 Compute 서비스를 재배포하려는 경우에 유용합니다.

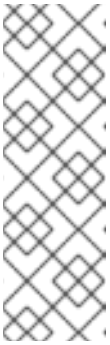
8. source 명령으로 언더클라우드 설정을 로드합니다.

```
(overcloud)$ source ~/stackrc
```

9. 스택에서 컴퓨팅 노드를 삭제합니다.

```
(undercloud)$ openstack overcloud node delete --stack <overcloud> <node>
```

- **<overcloud>**를 오버클라우드 스택의 이름 또는 UUID로 바꿉니다.
- **& lt;node >**를 삭제하려는 컴퓨팅 노드의 Compute 서비스 호스트 이름 또는 UUID로 바꿉니다.



참고

노드의 전원이 이미 꺼진 경우 이 명령은 **WARNING** 메시지를 반환합니다.

```
Ansible failed, check log at /var/lib/mistral/overcloud/ansible.log
WARNING: Scale-down configuration error. Manual cleanup of some
actions may be necessary. Continuing with node removal.
```

이 메시지는 무시해도 됩니다.

10. 오버클라우드 노드가 삭제될 때까지 기다립니다.

11. 노드 삭제가 완료되면 오버클라우드 스택의 상태를 확인합니다.

```
(undercloud)$ openstack stack list
```

표 12.3. 결과

상태	설명
UPDATE_COMPLETE	삭제 작업이 성공적으로 완료되었습니다.
UPDATE_FAILED	삭제 작업이 실패했습니다. 유지보수 모드에서 오버클라우드 노드가 삭제되지 않으면 하드웨어에 문제가 있을 수 있습니다.

12. 인스턴스 HA가 활성화된 경우 다음 작업을 수행합니다.

    a. 노드의 Pacemaker 리소스를 정리합니다.

    ```
    $ sudo pcs resource delete <scaled_down_node>
    $ sudo cibadmin -o nodes --delete --xml-text '<node id="<scaled_down_node>"/>'
    $ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
    <scaled_down_node>"/>'
    $ sudo cibadmin -o status --delete --xml-text '<node_state id="<scaled_down_node>"/>'
    $ sudo cibadmin -o status --delete-all --xml-text '<node id="<scaled_down_node>"/>' --
    force
    ```

    b. 노드의 STONITH 장치를 삭제합니다.

    ```
    $ sudo pcs stonith delete <device-name>
    ```

13. 오버클라우드에서 삭제된 Compute 노드를 교체하지 않는 경우 노드 수가 포함된 환경 파일의 **ComputeCount** 매개변수를 줄입니다. 일반적으로 이 파일의 이름은**node-info.yaml**로 지정됩니다. 예를 들어 한 노드를 삭제한 경우 노드 수를 4개의 노드에서 3개의 노드로 줄입니다.

    ```
    parameter_defaults:
      ...
      ComputeCount: 3
      ...
    ```

    노드 개수를 줄이면 **openstack overcloud deploy**를 실행할 때 director에서 새 노드를 프로비저닝하지 않습니다.

    오버클라우드 배포에서 삭제된 컴퓨팅 노드를 교체하는 경우 삭제된 컴퓨팅 노드 배치를참조하십시오.

12.3.2. 삭제된 Compute 노드 교체

오버클라우드 배포에서 삭제된 컴퓨팅 노드를 교체하려면 새 컴퓨팅 노드를 등록 및 검사하거나 삭제된 컴퓨팅 노드를 다시 추가할 수 있습니다. 또한 노드를 프로비저닝하도록 오버클라우드를 설정해야 합니다.

절차

1. 선택 사항: 삭제된 컴퓨팅 노드의 인덱스를 재사용하려면 컴퓨팅 노드가 제거될 때 거부 목록을 대체하도록 역할의 **removalPoliciesMode** 및 **removalPolicies** 매개변수를 구성합니다.

    ```
    parameter_defaults:
      <RoleName>RemovalPoliciesMode: update
      <RoleName>RemovalPolicies: [{'resource_list': []}]
    ```

2. 삭제된 컴퓨팅 노드를 교체합니다.

    - 새 컴퓨팅 노드를 추가하려면 새 노드를 등록, 검사, 태그를 지정하여 프로비저닝을 준비합니다. 자세한 내용은 기본 오버클라우드 설정을 참조하십시오.

    - 수동으로 제거한 컴퓨팅 노드를 다시 추가하려면 유지보수 모드에서 노드를 삭제합니다.

    ```
    (undercloud)$ openstack baremetal node maintenance unset <node_uuid>
    ```

3. 기존 오버클라우드를 배포하는 데 사용한 **openstack overcloud deploy** 명령을 재실행합니다.

4. 배포 프로세스가 완료될 때까지 기다립니다.

5. director가 새 컴퓨팅 노드를 성공적으로 등록했는지 확인합니다.

```
(undercloud)$ openstack baremetal node list
```

6. 역할 업데이트를 위해 Removed**PoliciesMode** 를 설정하기 위해 1단계를 수행한 경우, 컴퓨팅 노드가 제거될 때 현재 거부 목록에 해당 역할에 대한 Removed**PoliciesMode** 를 재설정하고, 를 추가하여 Compute 노드 인덱스를 현재 거부 목록에 추가해야 합니다.

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: append
```

7. 기존 오버클라우드를 배포하는 데 사용한 **openstack overcloud deploy** 명령을 재실행합니다.

12.4. CEPH STORAGE 노드 교체

director를 사용하여 director에서 생성한 클러스터에 있는 Ceph Storage 노드를 교체할 수 있습니다. 자세한 내용은 Deploying an Overcloud with Containerized Red Hat Ceph가이드를 참조하십시오.

12.5. OBJECT STORAGE 노드 교체

이 섹션에서는 클러스터를 원래 상태로 유지하면서 Object Storage 노드를 교체하는 방법을 설명합니다. 이 예제에서는 두 개의 노드로 이루어진 Object Storage 클러스터에서 **overcloud-objectstorage-1** 노드를 교체해야 합니다. 다음 절차의 목표는 한 개의 노드를 추가한 다음 **overcloud-objectstorage-1**을 삭제하여 노드를 효과적으로 교체하는 것입니다.

절차

1. **ObjectStorageCount** 매개변수를 사용하여 Object Storage 수를 늘립니다. 일반적으로 이 매개변수는 노드 수가 포함된 환경 파일인 **node-info.yaml**에 있습니다.

```
parameter_defaults:
  ObjectStorageCount: 4
```

ObjectStorageCount 매개변수는 해당 환경의 Object Storage 노드 수를 정의합니다. 이 예에서는 3개에서 4개로 노드를 확장합니다.

2. 업데이트된 **ObjectStorageCount** 매개변수를 사용하여 배포 명령을 실행합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates -e node-info.yaml
ENVIRONMENT_FILES
```

3. 배포 명령이 완료되면 오버클라우드에 추가 Object Storage 노드가 포함됩니다.

4. 데이터를 새 노드에 복제합니다. 노드(이 경우 **overcloud-objectstorage-1**)를 삭제하기 전에 *replication pass*가 새 노드에서 완료될 때까지 기다립니다./**var/log/swift/swift.log** 파일 전송 복제 진행 상태를 확인합니다. 전달이 완료되면 Object Storage 서비스에서 다음 예제와 비슷한 항목을 로그에 기록해야 합니다.

```
Mar 29 08:49:05 localhost object-server: Object replication complete.
Mar 29 08:49:11 localhost container-server: Replication run OVER
Mar 29 08:49:13 localhost account-server: Replication run OVER
```

5. 링에서 이전 노드를 삭제하려면 **ObjectStorageCount** 매개변수를 줄여 이전 노드를 생략합니다. 이 예제에서는 3으로 줄입니다.

```
parameter_defaults:
  ObjectStorageCount: 3
```

6. **remove-object-node.yaml**이라는 새 환경 파일을 생성합니다. 이 파일은 지정된 Object Storage 노드를 식별하고 삭제합니다. 다음 콘텐츠는 **overcloud-objectstorage-1**을 삭제하도록 지정합니다.

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

7. 배포 명령에 **node-info.yaml** 및 **remove-object-node.yaml** 파일을 모두 포함합니다.

```
(undercloud) $ openstack overcloud deploy --templates -e node-info.yaml
ENVIRONMENT_FILES -e remove-object-node.yaml
```

director가 오버클라우드에서 Object Storage 노드를 삭제하고 오버클라우드에서 나머지 노드를 업데이트하여 노드 삭제를 적용합니다.

12.6. 노드 블랙리스트 지정

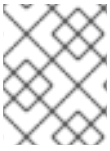
업데이트된 배포를 수신하지 못하도록 오버클라우드 노드를 제외할 수 있습니다. 이 기능은 코어 Heat 템플릿 컬렉션에서 업데이트된 매개변수 및 리소스 세트를 수신하지 못하도록 기존 노드를 제외한 상태에서 새 노드를 확장하려는 시나리오에서 유용합니다. 즉, 블랙리스트로 지정된 노드는 스택 작업의 영향을 받지 않습니다.

환경 파일에서 **DeploymentServerBlacklist** 매개변수를 사용하여 블랙리스트를 생성할 수 있습니다.

블랙리스트 설정

DeploymentServerBlacklist 매개변수는 서버 이름 목록입니다. 새 환경 파일을 작성하거나 기존 사용자 지정 환경 파일에 매개변수 값을 추가하고 파일을 배포 명령으로 전달합니다.

```
parameter_defaults:
  DeploymentServerBlacklist:
    - overcloud-compute-0
    - overcloud-compute-1
    - overcloud-compute-2
```



참고

매개변수 값의 서버 이름은 실제 서버 호스트 이름이 아니라 OpenStack Orchestration(heat)에 따른 이름입니다.

openstack overcloud deploy 명령을 사용하여 이 환경 파일을 추가합니다.

■

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e server-blacklist.yaml \
  [OTHER OPTIONS]
```

Heats는 업데이트된 Heat 배포를 수신하지 못하도록 목록의 모든 서버를 블랙리스트로 지정합니다. 스택 작업이 완료되면 블랙리스트로 지정된 서버는 변경되지 않고 그대로 유지됩니다. 작업 중에 os-collect-config 에이전트의 전원을 끄거나 중지할 수도 있습니다.

주의

- 노드를 블랙리스트로 지정할 때 주의하십시오. 요청된 변경 사항이 블랙리스트에서 어떻게 적용되는지 완전히 이해한 경우에만 블랙리스트를 사용하시기 바랍니다. 블랙리스트 기능을 사용하면 중단된 스택을 생성하거나 오버클라우드를 잘못 구성할 수 있습니다. 예를 들어 클러스터 구성 변경이 Pacemaker 클러스터의 모든 구성원에게 적용되는 경우 이러한 변경 중 Pacemaker 클러스터 구성원을 블랙리스트로 지정하면 클러스터가 실패할 수 있습니다.
- 업데이트 또는 업그레이드 절차 중에 블랙리스트 기능을 사용하지 마십시오. 이러한 절차에는 특정 서버에 대해 변경 사항을 분리하는 자체 방식이 있습니다. 자세한 내용은 *Upgrading Red Hat OpenStack Platform*문서를 참조하십시오.
- 블랙리스트에 서버를 추가할 때 블랙리스트에서 서버를 삭제할 때까지 해당 노드에 대한 추가 변경이 지원되지 않습니다. 업데이트, 업그레이드, 확장, 축소, 노드 교체 등이 이에 해당합니다. 예를 들어 새 컴퓨팅 노드로 오버클라우드를 확장하는 동안 기존 컴퓨팅 노드를 블랙리스트로 지정하면 블랙리스트로 지정된 노드는 **/etc/hosts** 및 **/etc/ssh/ssh_known_hosts**에 추가된 정보가 반영되지 않습니다. 이로 인해 대상 호스트에 따라 실시간 마이그레이션이 실패할 수 있습니다. 컴퓨팅 노드는 더 이상 블랙리스트로 지정되지 않는 다음 오버클라우드 배포 중에 **/etc/hosts** 및 **/etc/ssh/ssh_known_hosts**에 추가된 정보를 사용하여 업데이트됩니다. **/etc/hosts** 및 **/etc/ssh/ssh_known_hosts** 파일을 수동으로 수정하지 마십시오. **/etc/hosts** 및 **/etc/ssh/ssh_known_hosts** 파일을 수정하려면 *블랙리스트 삭제* 섹션에 설명된 대로 overcloud deploy 명령을 실행합니다.

블랙리스트 삭제

이후 스택 작업에 대해 블랙리스트를 지우려면 DeploymentServerBlacklist를 편집하여 빈 배열을 사용합니다.

```
parameter_defaults:
  DeploymentServerBlacklist: []
```

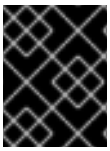
주의

DeploymentServerBlacklist 매개변수를 생략하지 마십시오. 매개변수를 생략하면 오버클라우드 배포에서 이전에 저장된 값을 사용합니다.

13장. 컨트롤러 노드 교체

특정 상황에서 고가용성 클러스터의 컨트롤러 노드에 오류가 발생할 수 있습니다. 이러한 경우 클러스터에서 해당 노드를 삭제하고 새 컨트롤러 노드로 교체해야 합니다.

컨트롤러 노드를 교체하려면 이 섹션에 있는 단계를 완료하십시오. 컨트롤러 노드 교체 프로세스에는 컨트롤러 노드 교체 요청으로 오버클라우드를 업데이트하는 **openstack overcloud deploy** 명령 실행 과정이 포함됩니다.



중요

다음 절차는 고가용성 환경에만 적용됩니다. 컨트롤러 노드를 하나만 사용하는 경우에는 다음 절차를 사용하지 마십시오.

13.1. 컨트롤러 교체 준비

오버클라우드 컨트롤러 노드를 교체하기 전에 Red Hat OpenStack Platform 환경의 현재 상태를 확인하는 것이 중요합니다. 현재 상태를 확인하면 컨트롤러 교체 프로세스 중에 복잡한 문제가 발생하는 것을 방지할 수 있습니다. 다음 사전 점검 목록을 사용하여 컨트롤러 노드 교체를 수행하는 것이 안전한지 확인합니다. 언더클라우드에서 검사 명령을 모두 실행합니다.

절차

1. 언더클라우드에서 **overcloud** 스택의 현재 상태를 확인합니다.

```
$ source stackrc
(undercloud) $ openstack stack list --nested
```

overcloud 스택 및 해당 하위 스택에 **CREATE_COMPLETE** 또는 **UPDATE_COMPLETE**가 있어야 합니다.

2. 언더클라우드 데이터베이스 백업을 수행합니다.

```
(undercloud) $ mkdir /home/stack/backup
(undercloud) $ sudo mysqldump --all-databases --quick --single-transaction | gzip >
/home/stack/backup/dump_db_undercloud.sql.gz
```

3. 새 노드를 프로비저닝할 때 언더클라우드에 이미지 캐싱 및 변환을 수행하는 데 필요한 10GB의 사용 가능한 스토리지가 있는지 확인합니다.

4. 새 컨트롤러 노드의 IP 주소를 재사용하는 경우 이전 컨트롤러에서 사용하는 포트를 삭제해야 합니다.

```
(undercloud) $ openstack port delete <port>
```

5. 실행 중인 컨트롤러 노드에서 Pacemaker의 상태를 확인합니다. 예를 들어 192.168.0.47이 실행 중인 컨트롤러 노드의 IP 주소인 경우 다음 명령을 사용하여 Pacemaker 상태 정보를 가져옵니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 'sudo pcs status'
```

출력에 기존 노드에서 실행 중인 서비스와 실패한 노드에서 중지된 모든 서비스가 표시됩니다.

6. 오버클라우드의 MariaDB 클러스터에 있는 각 노드에서 다음 매개변수를 확인합니다.

- **wsrep_local_state_comment: synced**
- **wsrep_cluster_size: 2**
  다음 명령을 사용하여 실행 중인 각 컨트롤러 노드에서 해당 매개변수를 확인합니다. 이 예제에서 컨트롤러 노드 IP 주소는 192.168.0.47 및 192.168.0.46입니다.

```
(undercloud) $ for i in 192.168.0.47 192.168.0.46 ; do echo "*** $i ***" ; ssh heat-admin@$i "sudo mysql -p\$(sudo hiera -c /etc/puppet/hiera.yaml mysql::server::root_password) --execute=\"SHOW STATUS LIKE 'wsrep_local_state_comment'; SHOW STATUS LIKE 'wsrep_cluster_size';\""; done
```

7. RabbitMQ 상태를 확인합니다. 예를 들어 192.168.0.47이 실행 중인 컨트롤러 노드의 IP 주소이면 다음 명령을 사용하여 상태 정보를 가져옵니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo docker exec \$(sudo docker ps -f name=rabbitmq-bundle -q) rabbitmqctl cluster_status"
```

   running_nodes 키는 사용 가능한 두 개의 노드만 표시하고, 실패한 노드는 표시하지 않습니다.

8. 펜싱이 활성화된 경우 비활성화합니다. 예를 들어 실행 중인 컨트롤러 노드의 IP 주소가 192.168.0.47인 경우 다음 명령을 사용하여 펜싱을 비활성화합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
```

   다음 명령을 사용하여 펜싱 상태를 확인합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
```

9. director 노드에서 **nova-compute** 서비스를 확인합니다.

```
(undercloud) $ sudo systemctl status openstack-nova-compute
(undercloud) $ openstack hypervisor list
```

   출력에 모든 유지보수 이외의 모드 노드가 **up**으로 표시됩니다.

10. 모든 언더클라우드 서비스가 실행 중인지 확인합니다.

```
(undercloud) $ sudo systemctl -t service
```

13.2. 백업 또는 스냅샷에서 컨트롤러 노드 복원

컨트롤러 노드가 실패하지만 물리적 디스크가 여전히 작동하는 경우 노드 자체를 교체하지 않고 백업 또는 스냅샷에서 노드를 복원할 수 있습니다.

중요

오류가 발생한 컨트롤러 노드에서 PXE 부팅에 사용되는 NIC의 MAC 주소가 디스크 교체 후에도 동일하게 유지되는지 확인합니다.

절차

- 컨트롤러 노드가 Red Hat Virtualization 노드이고 스냅샷을 사용하여 컨트롤러 노드를 백업하는 경우 스냅샷에서 노드를 복원합니다. 자세한 내용은 Red Hat Virtualization 가상 머신 관리 가이드의 "스냅샷을 사용하여 가상 머신 복원"을 참조하십시오.
- 컨트롤러 노드가 Red Hat Virtualization 노드이고 백업 스토리지 도메인을 사용하는 경우 백업 스토리지 도메인에서 노드를 복원합니다. 자세한 내용은 Red Hat Virtualization 관리 가이드의 "백업 스토리지 도메인을 사용하여 가상 머신 백업 및 복원"을 참조하십시오.
- Relax-and-Recover(ReaR) 툴에서 컨트롤러 노드의 백업 이미지가 있는 경우 ReaR 툴을 사용하여 노드를 복원합니다. 자세한 내용은 언더클라우드 및 컨트롤 플레인 백업 및 복원 가이드의 "컨트롤 플레인 복원"을 참조하십시오.
- 백업 또는 스냅샷에서 노드를 복구한 후 Galera 노드를 별도로 복구해야 할 수 있습니다. 자세한 내용은 How Galera works and how to rescue Galera clusters in the context of Red Hat OpenStack Platform 문서를 참조하십시오.
- 백업 복원을 완료한 후 필요한 모든 환경 파일과 함께 **openstack overcloud deploy** 명령을 실행하여 컨트롤러 노드 구성이 클러스터의 다른 노드의 구성과 일치하는지 확인합니다.
- 노드 백업이 없는 경우 표준 컨트롤러 교체 절차를 따라야 합니다.

13.3. CEPH MONITOR 데몬 삭제

스토리지 클러스터에서 **ceph-mon** 데몬을 삭제하려면 다음 절차를 수행합니다. 컨트롤러 노드가 Ceph 모니터 서비스를 실행하는 경우 다음 단계를 완료하여 ceph-mon 데몬을 삭제합니다. 다음 절차에서는 컨트롤러에 연결할 수 있다고 가정합니다.

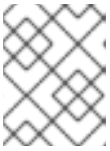
참고

클러스터에 새 컨트롤러 를 추가하면 새 Ceph 모니터 데몬도 자동으로 추가됩니다.

절차

1. 교체할 컨트롤러 에 연결하고 root로 전환합니다.

```
# ssh heat-admin@192.168.0.47
# sudo su -
```



참고

컨트롤러에 연결할 수 없는 경우 1단계와 2단계를 건너뛰고 작동하는 모든 컨트롤러 노드에서 3단계의 절차를 계속 진행합니다.

2. root로 모니터를 중지합니다.

```
# systemctl stop ceph-mon@<monitor_hostname>
```

예를 들면 다음과 같습니다.

```
# systemctl stop ceph-mon@overcloud-controller-1
```

3. 클러스터에서 모니터를 삭제합니다.

```
# ceph mon remove <mon_id>
```

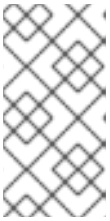
4. Ceph 모니터 노드에서 **/etc/ceph/ceph.conf** 의 monitor 항목을 제거합니다. 예를 들어 controller-1을 제거하면 controller-1의 IP 및 호스트 이름이 제거됩니다.
편집 전:

```
mon host = 172.18.0.21,172.18.0.22,172.18.0.24
mon initial members = overcloud-controller-2,overcloud-controller-1,overcloud-controller-0
```

편집 후:

```
mon host = 172.18.0.22,172.18.0.24
mon initial members = overcloud-controller-2,overcloud-controller-0
```

5. 다른 오버클라우드 노드의 **/etc/ceph/ceph.conf** 에 동일한 변경 사항을 적용합니다.



참고

대체 컨트롤러 노드를 추가하면 director가 **ceph.conf** 파일을 관련 오버클라우드 노드에서 업데이트합니다. 일반적으로 이 구성 파일은 director에서만 관리하며 수동으로 편집해서는 안 됩니다. 그러나 새 노드를 추가하기 전에 다른 노드를 재시작한 경우에는 파일을 직접 편집하여 일관성을 보장할 수 있습니다.

6. 선택적으로, 모니터 데이터를 압축하여 다른 서버에 아카이브를 저장할 수 있습니다.

```
# mv /var/lib/ceph/mon/<cluster>-<daemon_id> /var/lib/ceph/mon/removed-<cluster>-
<daemon_id>
```

13.4. 컨트롤러 교체를 위한 클러스터 준비

기존 노드를 교체하기 전에 Pacemaker가 더 이상 실행되지 않는지 확인한 후 Pacemaker 클러스터에서 해당 노드를 삭제해야 합니다.

절차

1. 컨트롤러 노드의 IP 주소 목록을 가져옵니다.

```
(undercloud) $ openstack server list -c Name -c Networks
+------------------------+------------------------+
| Name                   | Networks               |
+------------------------+------------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
| overcloud-controller-1 | ctlplane=192.168.0.45 |
| overcloud-controller-2 | ctlplane=192.168.0.46 |
+------------------------+------------------------+
```

2. 기존 노드에 계속 연결할 수 있는 경우 나머지 노드 중 하나에 로그인하여 기존 노드에서 Pacemaker를 중지합니다. 이 예제에서는 overcloud-controller-1의 Pacemaker를 중지합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs status | grep -w Online | grep -w
overcloud-controller-1"
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster stop overcloud-controller-1"
```

참고

기존 노드를 실제로 사용할 수 없거나 중지된 경우에는 해당 노드에서 Pacemaker가 이미 중지되었으므로 위 작업을 수행하지 않아도 됩니다.

3. **기존 노드에서 Pacemaker를 중지한 후(예: pcs status에 Stopped 로 표시됨) 각 노드의 corosync 구성에서 기존 노드를 삭제하고 Corosync를 다시 시작합니다. 이 예에서는 다음 명령은 overcloud-controller-0 에 로그인하고 overcloud-controller-2 는 노드를 제거합니다.**

```
(undercloud) $ for NAME in overcloud-controller-0 overcloud-controller-2; do IP=$(openstack
server list -c Networks -f value --name $NAME | cut -d "=" -f 2) ; ssh heat-admin@$IP "sudo
pcs cluster localnode remove overcloud-controller-1; sudo pcs cluster reload corosync"; done
```

4. **나머지 노드 중 하나에 로그인하고 crm_node 명령을 사용하여 클러스터에서 노드를 삭제합니다.**

```
(undercloud) $ ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ sudo crm_node -R overcloud-controller-1 --force
```

5. **오버클라우드 데이터베이스는 교체 절차 중 계속 실행되고 있어야 합니다. 다음 절차 중에 Pacemaker에서 Galera를 중지하지 않도록 하려면 실행 중인 컨트롤러 노드를 선택하고, 언더클라우드에서 컨트롤러 노드의 IP 주소를 사용하여 다음 명령을 실행합니다.**

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs resource unmanage galera-bundle"
```

13.5. 컨트롤러 노드 재사용

실패한 컨트롤러 노드를 재사용하고 새 노드로 재배포할 수 있습니다. 대체에 사용할 추가 노드가 없는 경우 이 방법을 사용합니다.

절차

1. **stackrc 파일을 소싱합니다.**

```
$ source ~/stackrc
```

2. **오버클라우드에서 오류가 발생한 노드의 연결을 끊습니다.**

```
$ openstack baremetal node undeploy <FAILED_NODE>
```

<FAILED_NODE>를 실패한 노드의 UUID로 바꿉니다. 이 명령은 OpenStack Compute(nova)의 오버클라우드 서버에서 OpenStack Bare Metal(ironic)의 노드의 연결을 끊습니다. 노드 정리를 활성화한 경우 이 명령은 노드 디스크에서 파일 시스템도 제거합니다.

3. **새 노드를 control 프로필로 태그합니다.**

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' <FAILED NODE>
```

4. 결함이 있는 디스크로 인해 컨트롤러 노드가 실패하면 이 시점에서 디스크를 교체하고 노드에서 인트로스펙션을 수행하여 새 디스크에서 인트로스펙션 데이터를 새로 고칠 수 있습니다.

```
$ openstack baremetal node manage <FAILED NODE>
$ openstack overcloud node introspect --all-manageable --provide
```

이제 오류가 발생한 노드가 노드를 교체 및 재배포할 준비가 되었습니다. 노드 교체를 수행할 때 실패한 노드는 새 노드 역할을 하며 증가된 인덱스를 사용합니다. 예를 들어 컨트롤 플레인 클러스터에 **overcloud-controller-0,overcloud-controller-1**, **overcloud-controller-2** 가 포함되어 있고**overcloud-controller-1** 을 새 노드로 재사용하면 새 노드 이름이 **overcloud-controller-3** 이 됩니다.

13.6. BMC IP 주소 재사용

실패한 컨트롤러 노드를 새 노드로 교체할 수 있지만 동일한 BMC IP 주소를 유지합니다. 실패한 노드를 제거하고, BMC IP 주소를 다시 할당하고, 새 노드를 새 baremetal 레코드로 추가하고, 인트로스펙션을 실행합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 오류가 발생한 노드를 삭제합니다.

```
$ openstack baremetal node undeploy <FAILED_NODE>
$ openstack baremetal node maintenance set <FAILED_NODE>
$ openstack baremetal node delete <FAILED_NODE>
```

<FAILED_NODE> 를 실패한 노드의 UUID로 바꿉니다. 큐에 이전 명령이 있는 경우**openstack baremetal node delete** 명령이 임시로 실패할 수 있습니다.**openstack baremetal node delete** 명령이 실패하면 이전 명령이 완료될 때까지 기다립니다. 이 작업을 수행하는 데 최대 5분이 걸릴 수 있습니다.

3. 오류가 발생한 노드의 BMC IP 주소를 새 노드에 할당합니다.

4. 새 노드를 새 baremetal 레코드로 추가합니다.

```
$ openstack overcloud node import newnode.json
```

오버클라우드 노드 등록에 대한 자세한 내용은 오버클라우드 노드 등록을 참조하십시오.

5. 새 노드에서 인트로스펙션을 수행합니다.

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

6. 연결되지 않은 노드를 나열하고 새 노드의 ID를 확인합니다.

```
$ openstack baremetal node list --unassociated
```

7. 새 노드를 **control** 프로필로 태그합니다.

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' <NEW NODE UUID>
```

13.7. 컨트롤러 노드 교체 트리거

기존 컨트롤러 노드를 삭제하고 새 컨트롤러 노드로 교체하려면 다음 단계를 완료합니다.

절차

1. **삭제하려는 노드의 UUID를 확인하고 NODEID 변수에 저장합니다.** ***NODE_NAME*****을 삭제하려는 노드 이름으로 교체해야 합니다.**

```
$ NODEID=$(openstack server list -f value -c ID --name NODE_NAME)
```

2. **Heat 리소스 ID를 식별하려면 다음 명령을 입력합니다.**

```
$ openstack stack resource show overcloud ControllerServers -f json -c attributes | jq --arg
NODEID "$NODEID" -c '.attributes.value | keys[] as $k | if .[$k] == $NODEID then "Node
index \($k) for \(.[$k])" else empty end'
```

3. **환경 파일 ~/templates/remove-controller.yaml을 생성하고 삭제하려는 컨트롤러 노드의 노드 인덱스를 포함합니다.**

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['NODE_INDEX']}]
```

4. **remove-controller.yaml 환경 파일과 함께 해당 환경과 관련된 기타 환경 파일을 포함하여 오버클라우드 배포 명령을 실행합니다.**

```
(undercloud) $ openstack overcloud deploy --templates \
    -e /home/stack/templates/remove-controller.yaml \
    [OTHER OPTIONS]
```

참고

배포 명령의 이 인스턴스에만 -e ~/templates/remove-controller.yaml을 포함합니다. 이후의 배포 작업에서는 이 환경 파일을 삭제합니다.

5. **director가 기존 노드를 삭제하고, 다음 노드 인덱스 ID를 사용하여 새 노드를 생성한 후 오버클라우드 스택을 업데이트합니다. 다음 명령을 사용하여 오버클라우드 스택의 상태를 확인할 수 있습니다.**

```
(undercloud) $ openstack stack list --nested
```

6. **배포 명령을 완료하면 director에 새 노드로 교체된 기존 노드가 표시됩니다.**

```
(undercloud) $ openstack server list -c Name -c Networks
+------------------------+-----------------------+
| Name                   | Networks              |
+------------------------+-----------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
```

```
| overcloud-controller-2 | ctlplane=192.168.0.46 |
| overcloud-controller-3 | ctlplane=192.168.0.48 |
+------------------------+-----------------------+
```

이제 새 노드에서 컨트롤 플레인 서비스를 실행합니다.

13.8. 컨트롤러 노드 교체 후 정리

노드 교체를 완료한 후에는 다음 단계를 완료하여 컨트롤러 클러스터를 종료합니다.

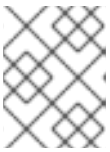
절차

1. 컨트롤러 노드에 로그인합니다.
2. Galera 클러스터의 Pacemaker 관리를 활성화하고 새 노드에서 Galera를 시작합니다.

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource refresh galera-bundle
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera-bundle
```

3. 최종 상태 검사를 수행하여 서비스가 올바르게 실행 중인지 확인합니다.

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



참고

서비스가 실패한 경우 **pcs resource refresh** 명령을 사용하여 문제를 해결한 후 실패한 서비스를 다시 시작합니다.

4. director를 종료합니다.

```
[heat-admin@overcloud-controller-0 ~]$ exit
```

5. 오버클라우드와 상호 작용할 수 있도록 source 명령으로 **overcloudrc** 파일을 로드합니다.

```
$ source ~/overcloudrc
```

6. 오버클라우드 환경의 네트워크 에이전트를 확인합니다.

```
(overcloud) $ openstack network agent list
```

7. 기존 노드의 에이전트가 표시되는 경우 삭제합니다.

```
(overcloud) $ for AGENT in $(openstack network agent list --host overcloud-controller-
1.localdomain -c ID -f value) ; do openstack network agent delete $AGENT ; done
```

8. 필요한 경우 새 노드의 L3 에이전트에 호스팅 라우터를 추가합니다. 다음 예제 명령을 사용하여 UUID 2d1c1dc1-d9d4-4fa9-b2c8-f29cd1a649d4를 사용하여 호스팅 라우터 r1을 L3 에이전트에 추가합니다.

```
(overcloud) $ openstack network agent add router -l3 2d1c1dc1-d9d4-4fa9-b2c8-
f29cd1a649d4 r1
```

9. **삭제된 노드의 Compute 서비스가 여전히 오버클라우드에 있으므로 삭제해야 합니다. Compute 서비스에서 삭제된 노드가 있는지 확인합니다.**

```
[stack@director ~]$ source ~/overcloudrc
(overcloud) $ openstack compute service list --host overcloud-controller-1.localdomain
```

10. **삭제된 노드의 Compute 서비스를 삭제합니다.**

```
(overcloud) $ for SERVICE in $(openstack compute service list --host overcloud-controller-
1.localdomain -c ID -f value ) ; do openstack compute service delete $SERVICE ; done
```

14장. 노드 재부팅

경우에 따라 언더클라우드 및 오버클라우드에서 노드를 재부팅해야 합니다. 다음 절차에서는 다양한 노드 유형을 재부팅하는 방법을 보여줍니다. 다음 참고 사항을 확인하십시오.

- 한 역할에 있는 모든 노드를 재부팅하는 경우 각 노드를 개별적으로 재부팅하는 것이 좋습니다. 이렇게 하면 재부팅 중 해당 역할에 대한 서비스를 유지하는 데 도움이 됩니다.
- OpenStack Platform 환경의 노드를 모두 재부팅하는 경우 다음 목록을 사용하여 재부팅 순서를 지정합니다.

권장되는 노드 재부팅 순서

1. 언더클라우드 노드 재부팅
2. 컨트롤러 노드 및 기타 구성 가능 노드 재부팅
3. 독립형 Ceph MON 노드 재부팅
4. Ceph Storage 노드 재부팅
5. Object Storage 서비스(swift) 노드 재부팅
6. 컴퓨팅 노드 재부팅

14.1. 언더클라우드 노드 재부팅

다음 절차에 따라 언더클라우드 노드를 재부팅합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. 언더클라우드를 재부팅합니다.

```
$ sudo reboot
```

3. 노드가 부팅될 때까지 기다립니다.

14.2. 컨트롤러 및 구성 가능 노드 재부팅

다음 절차는 구성 가능 역할을 기반으로 컨트롤러 노드 및 독립 실행형 노드를 재부팅합니다. 이렇게 하면 Compute 노드와 Ceph Storage 노드가 제외됩니다.

절차

1. 재부팅하려는 노드에 로그인합니다.
2. 선택 사항: 노드에서 Pacemaker 리소스를 사용하는 경우 클러스터를 중지합니다.

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

3. 노드를 재부팅합니다.

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

4. 노드가 부팅될 때까지 기다립니다.
5. 서비스를 확인합니다. 예를 들면 다음과 같습니다.
   a. 노드에서 Pacemaker 서비스를 사용하는 경우 노드가 클러스터에 다시 가입했는지 확인합니다.

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

   b. 노드에서 Systemd 서비스를 사용하는 경우 모든 서비스가 활성화되었는지 확인합니다.

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

   c. 모든 컨트롤러 및 구성 가능 노드에 대해 다음 단계를 반복합니다.

14.3. 독립형 CEPH MON 노드 재부팅

절차

1. Ceph MON 노드에 로그인합니다.
2. 노드를 재부팅합니다.

   ```
   $ sudo reboot
   ```

3. 노드가 부팅되고 MON 클러스터에 다시 참여할 때까지 기다립니다.

클러스터의 각 MON 노드에 대해 이 단계를 반복합니다.

14.4. CEPH STORAGE(OSD) 클러스터 재부팅

다음 절차에서는 Ceph Storage(OSD) 노드 클러스터를 재부팅합니다.

절차

1. Ceph MON 또는 컨트롤러 노드에 로그인하고 Ceph Storage 클러스터 재조정을 일시적으로 비활성화합니다.

   ```
   $ sudo ceph osd set noout
   $ sudo ceph osd set norebalance
   ```

2. 재부팅할 첫 번째 Ceph Storage 노드를 선택하고 로그인합니다.
3. 노드를 재부팅합니다.

   ```
   $ sudo reboot
   ```

4. 노드가 부팅될 때까지 기다립니다.
5. Ceph MON 또는 컨트롤러 노드에 로그인하고 클러스터 상태를 확인합니다.

```
$ sudo ceph -s
```

pgmap이 모든 **pgs**를 정상(**active+clean**)으로 보고하는지 확인합니다.

6. Ceph MON 또는 컨트롤러 노드에서 로그아웃하고, 다음 Ceph Storage 노드를 재부팅한 후 상태를 확인합니다. 모든 Ceph Storage 노드가 재부팅될 때까지 이 프로세스를 반복합니다.

7. 완료되면 Ceph MON 또는 컨트롤러 노드에 로그인하고 클러스터 재조정을 다시 활성화합니다.

```
$ sudo ceph osd unset noout
$ sudo ceph osd unset norebalance
```

8. 최종 상태 검사를 수행하여 클러스터가 **HEALTH_OK**를 보고하는지 확인합니다.

```
$ sudo ceph status
```

14.5. OBJECT STORAGE 서비스(SWIFT) 노드 재부팅

다음 절차에 따라 Object Storage 서비스(swift) 노드를 재부팅합니다. 클러스터의 모든 Object Storage 노드에 대해 다음 단계를 완료합니다.

절차

1. Object Storage 노드에 로그인합니다.
2. 노드를 재부팅합니다.

```
$ sudo reboot
```

3. 노드가 부팅될 때까지 기다립니다.
4. 클러스터의 각 Object Storage 노드에 대해 재부팅을 반복합니다.

14.6. 컴퓨팅 노드 재부팅

컴퓨팅 노드를 재부팅하려면 다음 워크플로우가 필요합니다.

- 새 인스턴스를 프로비저닝하지 않도록 재부팅할 컴퓨팅 노드를 선택하고 비활성화합니다.
- 인스턴스를 다른 컴퓨팅 노드로 마이그레이션하여 인스턴스 다운 타임을 최소화합니다.
- 빈 컴퓨팅 노드를 재부팅하고 활성화합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. 재부팅하려는 컴퓨팅 노드를 확인하려면 모든 컴퓨팅 노드를 나열합니다.

```
$ source ~/stackrc
(undercloud) $ openstack server list --name compute
```

3. 오버클라우드에서 컴퓨팅 노드를 선택하고 비활성화합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service list
(overcloud) $ openstack compute service set <hostname> nova-compute --disable
```

4. 컴퓨팅 노드에 모든 인스턴스를 나열합니다.

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

5. 인스턴스를 마이그레이션합니다. 마이그레이션 전략에 대한 자세한 내용은 컴퓨팅 노드 간 가상 머신 마이그레이션을 참조하십시오.

6. 컴퓨팅 노드에 로그인하고 재부팅합니다.

```
[heat-admin@overcloud-compute-0 ~]$ sudo reboot
```

7. 노드가 부팅될 때까지 기다립니다.

8. 컴퓨팅 노드를 활성화합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname> nova-compute --enable
```

9. 컴퓨팅 노드가 활성화되었는지 확인합니다.

```
(overcloud) $ openstack compute service list
```

15장. DIRECTOR 문제 해결

director 프로세스의 특정 단계에서 오류가 발생할 수 있습니다. 이 섹션에서는 일반적인 문제 진단을 위한 몇 가지 정보를 제공합니다.

director 구성 요소의 일반 로그를 참조하십시오.

- **/var/log** 디렉터리에는 많은 일반 OpenStack Platform 구성 요소에 대한 로그와 표준 Red Hat Enterprise Linux 애플리케이션에 대한 로그가 포함되어 있습니다.
- **journald** 서비스는 다양한 구성 요소에 대한 로그를 제공합니다. ironic은 두 개의 단위인 **openstack-ironic-api** 및 **openstack-ironic-conductor** 를 사용합니다. **ironic-inspector** 도 두 개의 단위를 사용합니다. **openstack-ironic-inspector** 및 **openstack-ironic-inspector-dnsmasq**. 각 구성 요소에 대해 두 유닛을 모두 사용합니다. 예를 들면 다음과 같습니다.

```
$ source ~/stackrc
(undercloud) $ sudo journalctl -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq
```

- **ironic-inspector** 는 또한 램디스크 로그를 **/var/log/ironic-inspector/ramdisk/** 에 gz로 압축된 tar 파일로 저장합니다. 파일 이름에는 노드의 날짜, 시간 및 IPMI 주소가 포함됩니다. 인트로스펙션 문제 진단에 이러한 로그를 사용하십시오.

15.1. 노드 등록 문제 해결

노드 등록 문제는 일반적으로 잘못된 노드 세부 정보 문제로 인해 발생합니다. 이 경우 **ironic** 을 사용하여 등록된 노드 데이터 문제를 해결합니다. 다음은 몇 가지 예입니다.

할당된 포트 UUID를 확인합니다.

```
$ source ~/stackrc
(undercloud) $ openstack baremetal port list --node [NODE UUID]
```

MAC 주소를 업데이트합니다.

```
(undercloud) $ openstack baremetal port set --address=[NEW MAC] [PORT UUID]
```

다음 명령을 실행합니다.

```
(undercloud) $ openstack baremetal node set --driver-info ipmi_address=[NEW IPMI ADDRESS] [NODE UUID]
```

15.2. 하드웨어 인트로스펙션 문제 해결

인트로스펙션 프로세스가 완료되어야 합니다. 그러나 검색 램디스크가 응답하지 않으면 ironic의 검색 데몬 (**ironic-inspector**)이 기본 1시간 후에 시간 초과됩니다. 경우에 따라 이는 검색 램디스크의 버그가 표시될 수 있지만, 일반적으로는 BIOS 부팅 설정 등 잘못된 환경 구성으로 인해 발생합니다.

다음은 잘못된 환경 구성이 발생하는 몇 가지 일반적인 시나리오와 진단 및 해결 방법에 대한 설명입니다.

시작 노드 인트로스펙션 시작 오류

일반적으로 인트로스펙션 프로세스는 **openstack overcloud node introspect** 명령을 사용합니다. 그러나 **ironic-inspector** 를 사용하여 인트로스펙션을 직접 실행하는 경우 검색 대상이 아닌 AVAILABLE 상태의 노드를 검색하지 못할 수 있습니다. 검색 전에 노드 상태를 MANAGEABLE 상태로 변경합니다.

```
$ source ~/stackrc
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

그런 다음 검색이 완료되면 프로비저닝 전에 다시 AVAILABLE로 변경합니다.

```
(undercloud) $ openstack baremetal node provide [NODE UUID]
```

검색 프로세스 중지

인트로스펙션 프로세스를 중지합니다.

```
$ source ~/stackrc
(undercloud) $ openstack baremetal introspection abort [NODE UUID]
```

프로세스가 시간 초과될 때까지 기다릴 수도 있습니다. 필요한 경우 **/etc/ironic-inspector/inspector.conf** 의 시간 제한 설정을 초 단위로 변경합니다.

인트로스펙션 램디스크에 액세스

인트로스펙션 램디스크는 동적 로그인 요소를 사용합니다. 즉, 인트로스펙션 디버깅 중에 노드에 액세스할 임시 암호 또는 SSH 키를 제공할 수 있습니다. 램디스크 액세스를 설정하려면 다음 프로세스를 사용하십시오.

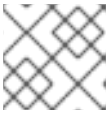
1. **openssl passwd -1** 명령에 임시 암호를 제공하여 MD5 해시를 생성합니다. 예를 들면 다음과 같습니다.

   ```
   $ openssl passwd -1 mytestpassword
   $1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
   ```

2. **/httpboot/inspector.ipxe** 파일을 편집하고 **kernel** 로 시작하는 행을 찾은 다음 **rootpwd** 매개변수와 MD5 해시를 추가합니다. 예를 들면 다음과 같습니다.

   ```
   kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-
   url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-hardware,logs
   systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1 ipa-inspection-
   benchmarks=cpu,mem,disk rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
   ```

   또는 공개 SSH 키를 사용하여 **sshkey** 매개변수를 추가할 수 있습니다.

   참고

   rootpwd 및 **sshkey** 매개변수에 모두 따옴표가 필요합니다.

3. 인트로스펙션을 시작하고 **arp** 명령 또는 DHCP 로그에서 IP 주소를 찾습니다.

   ```
   $ arp
   $ sudo journalctl -u openstack-ironic-inspector-dnsmasq
   ```

4. 임시 암호 또는 SSH 키를 사용하여 root 사용자로 SSH 연결을 수행합니다.

```
$ ssh root@192.168.24.105
```

인트로스펙션 스토리지 확인

director는 OpenStack Object Storage(swift)를 사용하여 인트로스펙션 프로세스 중에 가져온 하드웨어 데이터를 저장합니다. 이 서비스가 실행 중이 아니면 인트로스펙션이 실패할 수 있습니다. OpenStack Object Storage와 관련된 모든 서비스를 확인하여 서비스가 실행 중인지 확인합니다.

```
$ sudo systemctl list-units openstack-swift*
```

15.3. 워크플로우 및 실행 문제 해결

OpenStack Workflow(mistral) 서비스는 여러 OpenStack 작업을 워크플로우에 그룹화합니다. Red Hat OpenStack Platform은 이러한 워크플로우 세트를 사용하여 CLI 및 웹 UI에서 공통 기능을 수행합니다. 여기에는 베어 메탈 노드 제어, 검증, 계획 관리 및 오버클라우드 배포가 포함됩니다.

예를 들어 **openstack overcloud deploy** 명령을 실행할 때 OpenStack Workflow 서비스는 두 개의 워크플로우를 실행합니다. 첫 번째는 배포 계획을 업로드합니다.

```
Removing the current plan files
Uploading new plan files
Started Mistral Workflow. Execution ID: aef1e8c6-a862-42de-8bce-073744ed5e6b
Plan updated
```

두 번째는 오버클라우드 배포를 시작합니다.

```
Deploying templates in the directory /tmp/tripleoclient-LhRlHX/tripleo-heat-templates
Started Mistral Workflow. Execution ID: 97b64abe-d8fc-414a-837a-1380631c764d
2016-11-28 06:29:26Z [overcloud]: CREATE_IN_PROGRESS  Stack CREATE started
2016-11-28 06:29:26Z [overcloud.Networks]: CREATE_IN_PROGRESS  state changed
2016-11-28 06:29:26Z [overcloud.HeatAuthEncryptionKey]: CREATE_IN_PROGRESS  state
changed
2016-11-28 06:29:26Z [overcloud.ServiceNetMap]: CREATE_IN_PROGRESS  state changed
...
```

워크플로우 오브젝트

OpenStack Workflow는 다음 오브젝트를 사용하여 워크플로우를 추적합니다.

작업

관련 작업이 실행되면 OpenStack에서 수행하는 특정한 명령입니다. 예를 들면 쉘 스크립트 실행 또는 HTTP 요청 수행이 있습니다. 일부 OpenStack 구성 요소에는 OpenStack 워크플로우에서 사용하는 기본 제공 동작이 있습니다.

작업

실행할 동작 및 해당 동작을 실행한 결과를 정의합니다. 이러한 작업에는 일반적으로 동작 및 해당 동작과 연관된 다른 워크플로우가 있습니다. 작업이 완료되면 작업의 성공 또는 실패 여부에 따라 워크플로우에서 다른 작업을 지시합니다.

워크플로우

함께 그룹화되고 특정 순서로 실행되는 작업 집합입니다.

실행

특정 동작, 작업 또는 워크플로우 실행을 정의합니다.

워크플로우 오류 진단

OpenStack Workflow는 강력한 실행 기록 기능도 제공하므로 특정 명령이 실패했을 경우 문제를 식별할 수 있습니다. 예를 들어 워크플로우 실행이 실패하는 경우 문제 지점을 확인할 수 있습니다. 실패한 상태 **ERROR**가 있는 워크플로우 실행을 표시합니다.

```
$ source ~/stackrc
(undercloud) $ openstack workflow execution list | grep "ERROR"
```

실패한 워크플로우 실행의 UUID(예: dffa96b0-f679-4cd2-a490-4769a3825262)를 가져와서 실행 및 해당 출력을 확인합니다.

```
(undercloud) $ openstack workflow execution show dffa96b0-f679-4cd2-a490-4769a3825262
(undercloud) $ openstack workflow execution output show dffa96b0-f679-4cd2-a490-4769a3825262
```

이는 실행에서 실패한 작업에 대한 정보를 제공합니다. **openstack workflow execution show** 는 실행에 사용된 워크플로우(예: **tripleo.plan_management.v1.publish_ui_logs_to_swift**)도 표시합니다. 다음 명령을 사용하면 전체 워크플로우 정의를 볼 수 있습니다.

```
(undercloud) $ openstack workflow definition show
tripleo.plan_management.v1.publish_ui_logs_to_swift
```

이 명령은 워크플로우에서 특정 작업이 발생한 위치를 식별하는 데 유용합니다.

비슷한 명령 구문을 사용하여 작업 실행 및 해당 결과를 볼 수도 있습니다.

```
(undercloud) $ openstack action execution list
(undercloud) $ openstack action execution show 8a68eba3-0fec-4b2a-adc9-5561b007e886
(undercloud) $ openstack action execution output show 8a68eba3-0fec-4b2a-adc9-5561b007e886
```

이는 문제의 원인이 되는 특정 작업을 식별하는 데 유용합니다.

15.4. 오버클라우드 생성 문제 해결

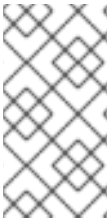
배포가 실패할 수 있는 세 가지 계층이 있습니다.

- 오케스트레이션(heat 및 nova 서비스)
- Bare Metal Provisioning(ironic 서비스)
- 배포 후 구성(Puppet)

이러한 수준에서 오버클라우드 배포에 실패한 경우 OpenStack 클라이언트 및 서비스 로그 파일을 사용하여 실패한 배포를 진단합니다. 다음 명령을 실행하여 실패 세부 정보를 표시할 수도 있습니다.

```
$ openstack stack failures list <OVERCLOUD_NAME> --long
```

<OVERCLOUD_NAME>을 해당 오버클라우드 이름으로 교체합니다.



참고

초기 오버클라우드 생성에 실패하면 **openstack stack delete overcloud** 명령을 사용하여 부분적으로 배포된 오버클라우드를 삭제하고 다시 시도할 수 있습니다. 이러한 초기 오버클라우드 생성이 실패한 경우에만 이 명령을 실행합니다. 완전히 배포된 운영 오버클라우드에서 이 명령을 실행하지 마십시오. 그렇지 않으면 전체 오버클라우드를 삭제합니다.

15.4.1. 배포 명령 내역 액세스

이전의 director 배포 명령 및 인수를 이해하면 문제 해결 및 지원에 유용할 수 있습니다. 이 정보는 **/home/stack/.tripleo/history** 에서 확인할 수 있습니다.

15.4.2. 오케스트레이션

대부분의 경우 Heat는 오버클라우드 생성에 실패한 후 실패한 오버클라우드 스택을 표시합니다.

```
$ source ~/stackrc
(undercloud) $ openstack stack list --nested --property status=FAILED
+----------------------+------------+--------------------+----------------------+
| id                   | stack_name | stack_status       | creation_time        |
+----------------------+------------+--------------------+----------------------+
| 7e88af95-535c-4a55... | overcloud  | CREATE_FAILED      | 2015-04-06T17:57:16Z |
+----------------------+------------+--------------------+----------------------+
```

스택 목록이 비어 있으면 초기 Heat 설정에 문제가 있음을 나타냅니다. Heat 템플릿 및 구성 옵션을 확인하고 **openstack overcloud deploy** 를 실행한 후 표시되는 오류 메시지를 확인합니다.

15.4.3. Bare Metal Provisioning

ironic 에서 등록된 모든 노드와 노드의 현재 상태를 확인합니다.

```
$ source ~/stackrc
(undercloud) $ openstack baremetal node list

+----------+------+---------------+-------------+-----------------+-------------+
| UUID     | Name | Instance UUID | Power State | Provision State | Maintenance |
+----------+------+---------------+-------------+-----------------+-------------+
| f1e261...| None | None          | power off   | available       | False       |
| f0b8c1...| None | None          | power off   | available       | False       |
+----------+------+---------------+-------------+-----------------+-------------+
```

다음은 프로비저닝 프로세스에서 발생하는 몇 가지 일반적인 문제입니다.

- 결과 테이블에서 프로비저닝 상태 및 유지보수 열을 검토합니다. 다음을 확인하십시오.
  - 빈 테이블 또는 예상보다 적은 노드
  - Maintenance가 True로 설정
  - 프로비저닝 상태가 **manageable** 로 설정되어 있습니다. 이는 일반적으로 등록 또는 검색 프로세스에 문제가 있음을 나타냅니다. 예를 들어 유지보수가 자동으로 True로 설정된 경우 일반적으로 노드에서 잘못된 전원 관리 인증 정보를 사용합니다.
- 프로비저닝 상태가 **available** 인 경우 베어 메탈 배포가 시작되기 전에 문제가 발생한 것입니다.

- 프로비저닝 상태가 **active** 이고 전원 상태가**power on** 인 경우 베어 메탈 배포가 성공적으로 완료된 것입니다. 즉, 배포 후 구성 단계에서 문제가 발생했습니다.
- 프로비저닝 상태가 노드에 대해 **wait call-back** 이면 베어 메탈 프로비저닝 프로세스가 이 노드에 대해 아직 완료되지 않았습니다. 이 상태가 변경될 때까지 기다립니다. 그렇지 않으면 오류가 발생한 노드의 가상 콘솔에 연결하여 출력을 확인합니다.
- 프로비저닝 상태가 **error** 또는 **deploy failed** 인 경우 베어 메탈 프로비저닝이 이 노드에 대해 실패했습니다. 베어 메탈 노드의 세부 사항을 확인합니다.

```
(undercloud) $ openstack baremetal node show [NODE UUID]
```

오류 설명이 포함된 **last_error** 필드를 찾습니다. 오류 메시지가 모호한 경우 로그를 사용하여 이를 명확히 지정할 수 있습니다.

```
(undercloud) $ sudo journalctl -u openstack-ironic-conductor -u openstack-ironic-api
```

- 대기 시간 초과 오류가 표시되고 노드 Power State가**power on** 인 경우 오류가 발생한 노드의 가상 콘솔에 연결하여 출력을 확인합니다.

15.4.4. 배포 후 구성

구성 단계에서 발생할 수 있는 작업은 많습니다. 예를 들어 설정 문제로 인해 특정 Puppet 모듈이 완료되지 못할 수 있습니다. 이 섹션에서는 이러한 문제를 진단하는 프로세스를 제공합니다.

오버클라우드 스택의 모든 리소스를 나열하여 실패한 리소스를 확인합니다.

```
$ source ~/stackrc
(undercloud) $ openstack stack resource list overcloud --filter status=FAILED
```

실패한 모든 리소스의 테이블이 표시됩니다.

실패한 리소스를 표시합니다.

```
(undercloud) $ openstack stack resource show overcloud [FAILED RESOURCE]
```

진단에 도움이 될 수 있는 **resource_status_reason** 필드의 모든 정보를 확인합니다.

nova 명령을 사용하여 오버클라우드 노드의 IP 주소를 확인합니다.

```
(undercloud) $ openstack server list
```

배포된 노드 중 하나에 **heat-admin** 사용자로 로그인합니다. 예를 들어 스택의 리소스 목록에 컨트롤러 노드에서 발생한 오류가 표시되면 컨트롤러 노드에 로그인합니다. **heat-admin** 사용자에게는 sudo 액세스 권한이 있습니다.

```
(undercloud) $ ssh heat-admin@192.168.24.14
```

실패 가능한 이유로 **os-collect-config** 로그를 확인합니다.

```
[heat-admin@overcloud-controller-0 ~]$ sudo journalctl -u os-collect-config
```

경우에 따라 nova가 노드 전체를 배포하지 못하는 경우가 있습니다. 이 상황은 오버클라우드 역할 유형 중 하나에 실패한 **OS::Heat::ResourceGroup** 으로 표시됩니다. **nova** 를 사용하여 이 경우 실패를 확인합니다.

```
(undercloud) $ openstack server list
(undercloud) $ openstack server show [SERVER ID]
```

표시된 가장 일반적인 오류는 오류 메시지 **No valid host**를 참조하십시오. 이 오류 문제 해결에 대한 자세한 내용은 15.6절. ""No Valid Host Found" 오류 문제 해결"을 참조하십시오. 다른 경우에는 추가 문제 해결을 위해 다음 로그 파일을 확인합니다.

- **/var/log/nova/***
- **/var/log/heat/***
- **/var/log/ironic/***

컨트롤러 노드의 배포 후 프로세스는 배포에 5개의 주요 단계를 사용합니다. 여기에는 다음이 포함됩니다.

표 15.1. 컨트롤러 노드 구성 단계

Step	설명
ControllerDeployment_Step1	Pacemaker, RabbitMQ, Memcached, Redis 및 Galera를 포함한 초기 로드 밸런싱 소프트웨어 구성.
ControllerDeployment_Step2	OpenStack Platform 서비스의 Pacemaker 구성, HAProxy, MongoDB, Galera, Ceph Monitor, 데이터베이스 초기화를 포함한 초기 클러스터 구성.
ControllerDeployment_Step3	OpenStack Object Storage의 초기 링 빌드(**swift**). 모든 OpenStack Platform 서비스 구성 (**nova**,**neutron**,**cinder**,**sahara**,**ceilometer**,**heat**,**horizon**,**aodh**,**gnocchi**).
ControllerDeployment_Step4	서비스 시작 순서 및 서비스 시작 매개 변수를 결정하는 제약 조건을 포함하여 Pacemaker에서 서비스 시작 설정을 구성합니다.
ControllerDeployment_Step5	OpenStack ID(**keystone**)의 프로젝트, 역할 및 사용자의 초기 구성.

15.5. 프로비저닝 네트워크에서 IP 주소 충돌 문제 해결

대상 호스트가 이미 사용 중인 IP 주소를 할당하면 검색 및 배포 작업이 실패합니다. 이 문제를 방지하려면 프로비저닝 네트워크에 대한 포트 스캔을 수행하여 검색 IP 범위 및 호스트 IP 범위를 사용할 수 있는지 확인합니다.

언더클라우드 호스트에서 다음 단계를 수행합니다.

nmap을 설치합니다.

```
$ sudo yum install nmap
```

nmap을 사용하여 활성 주소에 대한 IP 주소 범위를 스캔합니다. 이 예에서는 192.168.24.0/24 범위를 스캔하고, 이 범위를 프로비저닝 네트워크의 IP 서브넷으로 교체합니다(CIDR 비트마스크 표기 사용).

```
$ sudo nmap -sn 192.168.24.0/24
```

nmap 스캔 출력을 검토합니다.

예를 들어 언더클라우드의 IP 주소 및 서브넷에 있는 다른 호스트가 모두 표시되어야 합니다. 활성 IP 주소가 undercloud.conf의 IP 범위와 충돌하는 경우 IP 주소 범위를 변경하거나 IP 주소가 사용 가능하도록 설정한 후에 오버클라우드 노드를 인트로스펙션을 실행하거나 배포해야 합니다.

```
$ sudo nmap -sn 192.168.24.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
Nmap scan report for 192.168.24.1
Host is up (0.00057s latency).
Nmap scan report for 192.168.24.2
Host is up (0.00048s latency).
Nmap scan report for 192.168.24.3
Host is up (0.00045s latency).
Nmap scan report for 192.168.24.5
Host is up (0.00040s latency).
Nmap scan report for 192.168.24.9
Host is up (0.00019s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
```

15.6. "NO VALID HOST FOUND" 오류 문제 해결

/var/log/nova/nova-conductor.log에 다음 오류가 포함되는 경우가 있습니다.

```
NoValidHost: No valid host was found. There are not enough hosts available.
```

즉, nova 스케줄러에서 새 인스턴스 부팅에 적합한 베어 메탈 노드를 찾을 수 없었습니다. 이렇게 하면 일반적으로 nova에서 찾을 것으로 예상되는 리소스와 ironic에서 nova에 알리는 리소스가 일치하지 않습니다. 이 경우 다음을 확인하십시오.

1. 인트로스펙션이 성공했는지 확인합니다. 그렇지 않으면 각 노드에 필수 ironic 노드 속성이 포함되어 있는지 확인합니다. 각 노드에 대해 다음을 수행합니다.

   ```
   $ source ~/stackrc
   (undercloud) $ openstack baremetal node show [NODE UUID]
   ```

   properties JSON 필드에 **cpus**, **cpu_arch**, **memory_mb** 및 **local_gb** 키에 대한 올바른 값이 있는지 확인합니다.

2. 사용된 nova 플레이버가 필요한 노드 수에 대해 위의 ironic 노드 속성을 초과하지 않는지 확인합니다.

   ```
   (undercloud) $ openstack flavor show [FLAVOR NAME]
   ```

3. **openstack baremetal node list** 에 따라 사용 가능한 상태에 있는 노드가 충분한지 확인합니다. **manageable** 상태의 노드는 일반적으로 인트로스펙션 실패를 의미합니다.

4. 노드가 유지보수 모드가 아닌지 확인합니다. **openstack baremetal node list** 를 사용하여 확인합니다. 유지보수 모드로 자동 변경되는 노드는 전원 인증 정보가 잘못된 것입니다. 이를 확인한 다음 유지보수 모드를 제거합니다.

```
(undercloud) $ openstack baremetal node maintenance unset [NODE UUID]
```

5. **AHC(Automated Health Check)** 툴을 사용하여 자동 노드 태깅을 수행하는 경우 각 플레이버/프로필에 해당하는 노드가 충분한지 확인합니다. **openstack baremetal node show** 의 **properties** 필드에서 **capabilities** 키를 확인합니다. 예를 들어 Compute 역할에 태그된 노드에는 **profile:compute**가 포함되어 있어야 합니다.

6. 인트로스펙션 후 노드 정보가 ironic에서 nova로 반영되는 데 시간이 걸립니다. director는 일반적으로 이 도구를 사용합니다. 하지만 일부 단계를 수동으로 수행한 경우 nova에서 단시간 동안 노드를 사용하지 못할 수 있습니다. 다음 명령을 사용하여 시스템의 총 리소스를 확인합니다.

```
(undercloud) $ openstack hypervisor stats show
```

15.7. 오버클라우드 생성 후 문제 해결

오버클라우드 생성 후 나중에 특정 오버클라우드 작업을 수행할 수 있습니다. 예를 들어 사용 가능한 노드를 확장하거나 문제가 있는 노드를 교체할 수 있습니다. 이러한 작업을 수행할 때 특정 문제가 발생할 수 있습니다. 이 섹션에서는 생성 후 작업 실패 문제를 진단하고 해결하는 몇 가지 조언을 제공합니다.

15.7.1. 오버클라우드 스택 수정

director를 통해 오버클라우드 스택을 수정할 때 문제가 발생할 수 있습니다. 스택 수정의 예는 다음과 같습니다.

- 노드 확장
- 노드 제거
- 노드 교체

스택 수정은 요청된 노드 수의 가용성 확인, 추가 노드 프로비저닝 또는 기존 노드 제거 후 Puppet 구성을 적용한다는 점에서 스택 생성 프로세스와 유사합니다. 다음은 오버클라우드 스택을 수정할 때 따라야 하는 몇 가지 지침입니다.

초기 단계로 15.4.4절. “배포 후 구성” 의 조언을 따르십시오. 이러한 단계를 수행하면오버클라우드 heat 스택 업데이트 관련 문제를 진단할 수 있습니다. 특히 다음 명령을 사용하여 문제가 있는 리소스를 식별합니다.

openstack stack list --show-nested

모든 스택을 나열합니다. **--show-nested** 는 모든 하위 스택과 해당 상위 스택을 표시합니다. 이 명령은 스택이 실패한 지점을 식별하는 데 도움이 됩니다.

OpenStack stack resource list overcloud

오버클라우드 스택의 모든 리소스와 해당 현재 상태를 나열합니다. 이를 통해 스택에서 오류가 발생하는 리소스를 식별하는 데 도움이 됩니다. heat 템플릿 컬렉션 및 Puppet 모듈의 각 매개변수 및 구성에 대한 이 리소스를 추적할 수 있습니다.

OpenStack 스택 이벤트 목록 오버클라우드

오버클라우드 스택과 관련된 모든 이벤트를 시간순으로 나열합니다. 여기에는 스택의 모든 리소스의 시작, 완료 및 실패가 포함됩니다. 이는 리소스 오류 지점을 식별하는 데 도움이 됩니다.

다음 몇 섹션에서는 특정 노드 유형에 대한 문제 진단 방법을 제공합니다.

15.7.2. Controller 서비스 오류

오버클라우드 컨트롤러 노드에는 대부분의 Red Hat OpenStack Platform 서비스가 포함되어 있습니다. 마찬가지로 고가용성 클러스터에서 여러 Controller 노드를 사용할 수 있습니다. 노드의 특정 서비스에 문제가 발생하면 고가용성 클러스터에서 특정 수준의 페일오버를 제공합니다. 그러나 오버클라우드가 전체 용량으로 작동하도록 결함이 있는 서비스를 진단해야 합니다.

컨트롤러 노드는 Pacemaker를 사용하여 고가용성 클러스터에서 리소스 및 서비스를 관리합니다. Pacemaker 구성 시스템(**pcs**) 명령은 Pacemaker 클러스터를 관리하는 툴입니다. 클러스터의 컨트롤러 노드에서 이 명령을 실행하여 구성 및 모니터링 기능을 수행합니다. 다음은 고가용성 클러스터에서 오버클라우드 서비스 문제를 해결하는 데 도움이 되는 몇 가지 명령입니다.

pcs status

활성화된 리소스, 실패한 리소스 및 온라인 노드를 포함하여 전체 클러스터에 대한 상태 개요를 제공합니다.

pcs resource show

리소스 목록 및 해당 노드를 표시합니다.

pcs resource disable [resource]

특정 리소스를 중지합니다.

pcs resource enable [resource]

특정 리소스를 시작합니다.

pcs cluster standby [node]

노드를 대기 모드로 전환합니다. 노드를 클러스터에서 더 이상 사용할 수 없습니다. 클러스터에 영향을 주지 않고 특정 노드에서 유지보수를 수행하는 데 유용합니다.

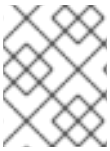
pcs cluster unstandby [node]

대기 모드에서 노드를 제거합니다. 이 노드는 클러스터에서 다시 사용할 수 있게 됩니다.

이러한 Pacemaker 명령을 사용하여 문제가 있는 구성 요소 및/또는 노드를 식별합니다. 구성 요소를 식별한 후 **/var/log/** 에서 해당 구성 요소 로그 파일을 봅니다.

15.7.3. 컨테이너화된 서비스 오류

오버클라우드 배포 중 또는 이후에 컨테이너화된 서비스가 실패하면 다음 권장 사항을 사용하여 실패의 근본 원인을 확인합니다.



참고

이러한 명령을 실행하기 전에 오버클라우드 노드에 로그인하고 언더클라우드에서 이러한 명령을 실행하지 않았는지 확인합니다.

컨테이너 로그 확인

각 컨테이너는 메인 프로세스의 표준 출력을 유지합니다. 이 출력은 컨테이너 실행 중에 실제로 수행되는 작업을 결정하는 데 도움이 되는 로그 역할을 합니다. 예를 들어 **keystone** 컨테이너의 로그를 보려면 다음 명령을 사용합니다.

```
$ sudo docker logs keystone
```

대부분의 경우 이 로그는 컨테이너 오류의 원인을 제공합니다.

컨테이너 검사

컨테이너 정보를 확인해야 하는 경우가 있습니다. 예를 들어 다음 명령을 사용하여 **keystone** 컨테이너 데이터를 확인합니다.

```
$ sudo docker inspect keystone
```

이는 낮은 수준의 구성 데이터를 포함하는 JSON 오브젝트를 제공합니다. 출력을 **jq** 명령으로 전달하여 특정 데이터를 구문 분석할 수 있습니다. 예를 들어 **keystone** 컨테이너에 대한 컨테이너 마운트를 보려면 다음 명령을 실행합니다.

```
$ sudo docker inspect keystone | jq .[0].Mounts
```

또한 **--format** 옵션을 사용하여 단일 행에 대한 데이터를 구문 분석할 수 있으며 이는 컨테이너 데이터 세트에 대해 명령을 실행할 때 유용합니다. 예를 들어 **keystone** 컨테이너 실행에 사용된 옵션을 재생성하려면 **--format** 옵션과 함께 다음 **inspect** 명령을 사용합니다.

```
$ sudo docker inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}' keystone
```

참고

--format 옵션은 쿼리를 생성할 때 Go 구문을 사용합니다.

이러한 옵션을 **docker run** 명령과 함께 사용하여 문제 해결을 위해 컨테이너를 다시 생성합니다.

```
$ OPTIONS=$( sudo docker inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo docker run --rm $OPTIONS /bin/bash
```

컨테이너에서 명령 실행

특정 Bash 명령을 통해 컨테이너 내부 정보를 가져와야 하는 경우가 있습니다. 이 경우 **docker** 명령을 사용하여 실행 중인 컨테이너 내에서 명령을 실행합니다. 예를 들어 **keystone** 컨테이너에서 명령을 실행하는 방법은 다음과 같습니다.

```
$ sudo docker exec -ti keystone <COMMAND>
```

참고

-ti 옵션은 대화식 가상 터미널(pseudoterminal)에서 명령을 실행합니다.

<COMMAND> 를 원하는 명령으로 바꿉니다. 예를 들어 각 컨테이너에는 서비스 연결을 확인하기 위한 상태 점검 스크립트가 있습니다. 다음 명령을 사용하여 **keystone**에 대해 상태 점검 스크립트를 실행할 수 있습니다.

```
$ sudo docker exec -ti keystone /openstack/healthcheck
```

컨테이너의 쉘에 액세스하려면 **/bin/bash** 를 명령으로 사용하여 **docker exec** 를 실행합니다.

```
$ sudo docker exec -ti keystone /bin/bash
```

컨테이너 내보내기

컨테이너가 실패하면 파일의 전체 콘텐츠를 확인해야 합니다. 이 경우 컨테이너의 전체 파일 시스템을 **tar** 아카이브로 내보낼 수 있습니다. 예를 들어 **keystone** 컨테이너 파일 시스템을 내보내려면 다음 명령을 실행합니다.

```
$ sudo docker export keystone -o keystone.tar
```

이 명령은 **keystone.tar** 아카이브를 생성하여 추출 및 확인할 수 있습니다.

15.7.4. Compute 서비스 오류

컴퓨팅 노드는 Compute 서비스를 사용하여 하이퍼바이저 기반 작업을 수행합니다. 따라서 컴퓨팅 노드에 대한 주요 진단은 이 서비스와 관련이 있습니다. 예를 들면 다음과 같습니다.

- 컨테이너 상태를 표시합니다.

  ```
  $ sudo docker ps -f name=nova_compute
  ```

- 컴퓨팅 노드의 주요 로그 파일은 **/var/log/containers/nova/nova-compute.log**입니다. 컴퓨팅 노드 통신에 문제가 발생하면 일반적으로 이 로그 파일을 사용하여 진단을 시작하는 것이 좋습니다.

- 컴퓨팅 노드에서 유지보수를 수행할 때 기존 인스턴스를 호스트에서 운영 가능한 컴퓨팅 노드로 마이그레이션한 다음 해당 노드를 비활성화합니다. 자세한 내용은 Migrating virtual machine instances between Compute nodes를 참조하십시오.

15.7.5. Ceph Storage 서비스 오류

Red Hat Ceph Storage 클러스터에 발생하는 문제는 *Red Hat Ceph Storage Configuration Guide*의 "Logging Configuration Reference"를 참조하십시오. 이 섹션에서는 모든 Ceph 스토리지 서비스에 대한 진단 로그 정보를 제공합니다.

15.8. 언더클라우드 튜닝

이 섹션의 지침은 언더클라우드의 성능을 향상시키는 데 도움이 됩니다. 필요에 따라 권장 사항을 구현합니다.

- Identity 서비스(keystone)는 다른 OpenStack 서비스에 대한 액세스 제어에 토큰 기반 시스템을 사용합니다. 일정한 기간이 지나면 데이터베이스에는 사용하지 않는 많은 토큰이 누적됩니다. 기본 cronjob은 매일 토큰 테이블을 플러시합니다. 환경을 모니터링하고 토큰 플러시 간격을 필요에 따라 조정하는 것이 좋습니다. 언더클라우드의 경우 **crontab -u keystone -e** 를 사용하여 간격을 조정할 수 있습니다. 이는 일시적인 변경 사항이며 **openstack undercloud update** 에서 이 cronjob을 다시 기본값으로 재설정합니다.

- Heat는 **openstack overcloud deploy** 를 실행할 때마다 모든 템플릿 파일의 사본을 데이터베이스의 **raw_template** 테이블에 저장합니다. **raw_template** 테이블은 과거의 모든 템플릿을 유지하고 크기를 늘립니다. **raw_templates** 테이블에서 사용되지 않는 템플릿을 제거하려면 매일 cronjob을 생성하여 데이터베이스에 존재하는 사용되지 않는 템플릿을 하루 이상 지웁니다.

  ```
  0 04 * * * /bin/heat-manage purge_deleted -g days 1
  ```

- **openstack-heat-engine** 및 **openstack-heat-api** 서비스가 너무 많은 리소스를 사용하는 경우가 있습니다. 이 경우 **/etc/heat/heat.conf** 에서 **max_resources_per_stack=-1** 을 설정하고 heat 서비스를 다시 시작합니다.

```
$ sudo systemctl restart openstack-heat-engine openstack-heat-api
```

- director에 동시 노드 프로비저닝을 수행하기에 충분한 리소스가 없을 수 있습니다. 기본값은 동시에 10개 노드입니다. 동시 노드 수를 줄이려면 **/etc/nova/nova.conf** 의 **max_concurrent_builds** 매개변수를 10보다 작은 값으로 설정하고 nova 서비스를 다시 시작합니다.

```
$ sudo systemctl restart openstack-nova-api openstack-nova-scheduler
```

- **/etc/my.cnf.d/galera.cnf** 파일을 편집합니다. 조정의 몇 가지 권장 값은 다음과 같습니다.

  max_connections

  데이터베이스에 대한 동시 연결 수입니다. 권장 값은 4096입니다.

  innodb_additional_mem_pool_size

  데이터베이스에서 데이터 사전 정보 및 기타 내부 데이터 구조를 저장하는 데 사용하는 메모리 풀의 크기(바이트)입니다. 언더클라우드에 대한 기본값은 일반적으로 8M이며 이상적인 값은 20M입니다.

  innodb_buffer_pool_size

  데이터베이스에서 테이블 및 인덱스 데이터를 캐시하는 메모리 영역의 버퍼 풀 크기(바이트)입니다. 언더클라우드에 대한 기본값은 일반적으로 128M이며, 이상적인 값은 1000M입니다.

  innodb_flush_log_at_trx_commit

  커밋 관련 I/O 작업을 다시 정렬하고 배치로 수행할 때 가능한 고성능과 커밋 작업을 위한 엄격한 ACID 규정 준수 간의 균형을 제어합니다. 1로 설정합니다.

  innodb_lock_wait_timeout

  데이터베이스 트랜잭션이 종료되기 전에 행 잠금을 기다리는 시간(초)입니다. 50으로 설정합니다.

  innodb_max_purge_lag

  이 변수는 제거 작업이 지연될 때 INSERT, UPDATE 및 DELETE 작업을 지연하는 방법을 제어합니다. 10000으로 설정합니다.

  innodb_thread_concurrency

  동시 운영 체제 스레드 제한입니다. 각 CPU 및 디스크 리소스에 대해 적어도 두 개의 스레드를 제공하는 것이 좋습니다. 예를 들어 쿼드 코어 CPU와 단일 디스크를 사용하는 경우 10개의 스레드를 사용합니다.

- heat에 오버클라우드 생성을 수행할 충분한 작업자가 있는지 확인합니다. 일반적으로 언더클라우드에 있는 CPU 수에 따라 다릅니다. 작업자 수를 수동으로 설정하려면 **/etc/heat/heat.conf** 파일을 편집하고, **num_engine_workers** 매개변수를 필요한 작업자 수(대부분 4)로 설정한 후 heat 엔진을 다시 시작합니다.

```
$ sudo systemctl restart openstack-heat-engine
```

15.9. SOSREPORT 생성

OpenStack Platform에 대한 지원을 받기 위해 Red Hat에 문의하려는 경우 **sosreport**를 생성해야 할 수도 있습니다. **sosreport** 생성 방법에 대한 자세한 내용은 다음 지식베이스 문서를 참조하십시오.

- "How to collect all required logs for Red Hat Support to investigate an OpenStack issue"

15.10. 언더클라우드 및 오버클라우드 관련 중요 로그

문제를 해결할 때 다음 로그를 사용하여 언더클라우드 및 오버클라우드에 대한 정보를 찾습니다.

표 15.2. 언더클라우드 관련 중요 로그

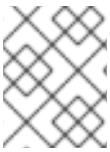
정보	로그 위치
OpenStack Compute 로그	**/var/log/nova/nova-compute.log**
OpenStack Compute API 상호 작용	**/var/log/nova/nova-api.log**
OpenStack Compute Conductor 로그	**/var/log/nova/nova-conductor.log**
OpenStack Orchestration 로그	**/var/log/heat/heat-engine.log**
OpenStack Orchestration API 상호 작용	**/var/log/heat/heat-api.log**
OpenStack Orchestration CloudFormations 로그	**/var/log/heat/heat-api-cfn.log**
OpenStack Bare Metal Conductor 로그	**/var/log/ironic/ironic-conductor.log**
OpenStack Bare Metal API 상호 작용	**/var/log/ironic/ironic-api.log**
인트로스펙션	**/var/log/ironic-inspector/ironic-inspector.log**
OpenStack Workflow Engine 로그	**/var/log/mistral/engine.log**
OpenStack Workflow Executor 로그	**/var/log/mistral/executor.log**
OpenStack Workflow API 상호 작용	**/var/log/mistral/api.log**

표 15.3. 오버클라우드 관련 중요 로그

정보	로그 위치
Cloud-Init 로그	**/var/log/cloud-init.log**
오버클라우드 구성(마지막 Puppet 실행 요약)	**/var/lib/puppet/state/last_run_summary.yaml**
오버클라우드 구성(마지막 Puppet 실행 보고서)	**/var/lib/puppet/state/last_run_report.yaml**
오버클라우드 구성(전체 Puppet 보고서)	**/var/lib/puppet/reports/overcloud-*/***
오버클라우드 구성(각 Puppet 실행의 stdout)	**/var/run/heat-config/deployed/*-stdout.log**
오버클라우드 설정(각 Puppet 실행의stderr)	**/var/run/heat-config/deployed/*-stderr.log**
고가용성 로그	**/var/log/pacemaker.log**

부록 A. SSL/TLS 인증서 구성

공용 끝점에서의 통신에 SSL/TLS를 사용하도록 언더클라우드를 설정할 수 있습니다. 그러나 자체 인증 기관의 SSL 인증서를 사용하는 경우 인증서에 다음 섹션의 구성 단계가 필요합니다.



참고

오버클라우드 SSL/TLS 인증서 생성의 경우 *Advanced Overcloud Customization* 가이드의 "Enabling SSL/TLS on Overcloud Public Endpoints"를 참조하십시오.

A.1. 서명 호스트 초기화

서명 호스트는 새 인증서를 생성하고 인증 기관을 통해 서명하는 호스트입니다. 선택한 서명 호스트에서 SSL 인증서를 생성한 적이 없는 경우 새 인증서에 서명할 수 있도록 호스트를 초기화해야 할 수 있습니다.

/etc/pki/CA/index.txt 파일은 서명된 모든 인증서의 레코드를 저장합니다. 이 파일이 있는지 확인합니다. 없는 경우 빈 파일을 생성합니다.

```
$ sudo touch /etc/pki/CA/index.txt
```

/etc/pki/CA/serial 파일은 서명할 다음 인증서에 사용할 다음 일련번호를 식별합니다. 이 파일이 있는지 확인합니다. 존재하지 않는 경우 새 시작 값으로 새 파일을 생성합니다.

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

A.2. 인증 기관 생성

일반적으로는 외부 인증 기관을 통해 SSL/TLS 인증서에 서명합니다. 고유한 인증 기관을 사용하려는 경우도 있습니다. 예를 들어 내부 전용 인증 기관을 사용하도록 설정할 수 있습니다.

예를 들어 인증 기관 역할을 할 키와 인증서 쌍을 생성합니다.

```
$ sudo openssl genrsa -out ca.key.pem 4096
$ sudo openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

openssl req 명령은 기관에 대한 특정 세부 정보를 요청합니다. 이러한 세부 사항을 입력합니다.

이렇게 하면 **ca.crt.pem** 이라는 인증 기관 파일이 생성됩니다.

A.3. 클라이언트에 인증 기관 추가

외부 클라이언트가 SSL/TLS를 사용하여 통신하려는 경우 Red Hat OpenStack Platform 환경에 액세스해야 하는 각 클라이언트에 인증 기관 파일을 복사합니다. 클라이언트에 복사되면 클라이언트에서 다음 명령을 실행하여 인증 기관 신뢰 번들에 추가합니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

A.4. SSL/TLS 키 생성

언더클라우드 또는 오버클라우드 인증서를 생성할 다른 시점에서 사용하는 SSL/TLS 키(**server.key.pem**)를 생성하려면 다음 명령을 실행합니다.

```
$ openssl genrsa -out server.key.pem 2048
```

A.5. SSL/TLS 인증서 서명 요청 생성

다음 절차에서는 언더클라우드 또는 오버클라우드에 대한 인증서 서명 요청을 생성합니다.

사용자 정의할 기본 OpenSSL 구성 파일을 복사합니다.

```
$ cp /etc/pki/tls/openssl.cnf .
```

사용자 정의 **openssl.cnf** 파일을 편집하고 director에 사용할 SSL 매개변수를 설정합니다. 수정할 매개변수 유형의 예제는 다음과 같습니다.

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

commonName_default 를 다음 중 하나로 설정합니다.

- IP 주소를 사용하여 SSL/TLS를 통해 액세스하는 경우 **undercloud.conf** 에서 **undercloud_public_host** 매개변수를 사용합니다.
- 정규화된 도메인 이름을 사용하여 SSL/TLS를 통해 액세스하는 경우 도메인 이름을 대신 사용합니다.

subjectAltName = @alt_names 를 **v3_req** 섹션에 추가합니다.

alt_names 섹션을 편집하여 다음 항목을 포함합니다.

- **IP** - 클라이언트가 SSL을 통해 director에 액세스하는 IP 주소 목록입니다.
- **DNS** - 클라이언트가 SSL을 통해 director에 액세스하는 도메인 이름 목록입니다. 또한 공용 API IP 주소를 **alt_names** 섹션 끝에 DNS 항목으로 추가합니다.

openssl.cnf 에 대한 자세한 내용을 보려면**man openssl.cnf** 를 실행합니다.

다음 명령을 실행하여 인증서 서명 요청(**server.csr.pem**)을 생성합니다.

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

-key 옵션에 대해 A.4절. "SSL/TLS 키 생성" 에서 생성한 SSL/TLS 키를 포함해야 합니다.

다음 섹션에서 **server.csr.pem** 파일을 사용하여 SSL/TLS 인증서를 생성합니다.

A.6. SSL/TLS 인증서 생성

다음 명령은 언더클라우드 또는 오버클라우드에 대한 인증서를 생성합니다.

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

이 명령은 다음을 사용합니다.

- v3 확장을 지정하는 구성 파일입니다. 이를 **-config** 옵션으로 추가합니다.
- 인증서를 생성하고 인증 기관을 통해 서명하는 A.5절. "SSL/TLS 인증서 서명 요청 생성"의 인증서 서명 요청입니다. 이를 **-in** 옵션으로 추가합니다.
- A.2절. "인증 기관 생성" 에서 생성한 인증 기관. 이 옵션을**-cert** 옵션으로 추가합니다.
- A.2절. "인증 기관 생성" 에서 생성한 인증 기관 개인 키입니다. 이 키를**-keyfile** 옵션으로 추가합니다.

이렇게 하면 **server.crt.pem** 이라는 인증서가 생성됩니다. 이 인증서를A.4절. "SSL/TLS 키 생성" 의 SSL/TLS 키와 함께 사용하여 SSL/TLS를 활성화합니다.

A.7. 언더클라우드에서 인증서 사용

다음 명령을 실행하여 인증서와 키를 함께 결합합니다.

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

이렇게 하면 **undercloud.pem** 파일이 생성됩니다. **undercloud.conf** 파일의 **undercloud_service_certificate** 옵션에 이 파일의 위치를 지정합니다. 이 파일에는 HAProxy 툴에서 읽을 수 있도록 특수 SELinux 컨텍스트도 필요합니다. 다음 예제를 가이드로 사용합니다.

```
$ sudo mkdir /etc/pki/instack-certs
$ sudo cp ~/undercloud.pem /etc/pki/instack-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/instack-certs(/.*)?"
$ sudo restorecon -R /etc/pki/instack-certs
```

undercloud.pem 파일 위치를 **undercloud.conf** 파일의 **undercloud_service_certificate** 옵션에 추가합니다. 예를 들면 다음과 같습니다.

```
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
```

또한 A.2절. “인증 기관 생성” 의 인증 기관을 언더클라우드의 신뢰할 수 있는 인증 기관 목록에 추가하여 언더클라우드 내의 다른 서비스에서 인증 기관에 액세스할 수 있도록 합니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

언더클라우드가 이미 설치되어 있고 **openstack undercloud install** 을 실행하여 기존 설정을 업데이트하려는 경우 **haproxy** 서비스를 다시 시작하여 설정을 다시 로드해야 합니다.

```
$ sudo systemctl restart haproxy
```

4.8절. “Director 설정 ” 의 지침에 따라 언더클라우드를 계속 설치합니다.

부록 B. 전원 관리 드라이버

director는 기본적으로 전원 관리 컨트롤에 IPMI를 사용하지만 다른 전원 관리 유형도 지원합니다. 이 부록은 지원되는 전원 관리 기능 목록을 제공합니다. 6.1절. "오버클라우드에 노드 등록"에 대해 이러한 전원 관리 설정을 사용하십시오.

B.1. REDFISH

DMTF(Distributed Management Task Force)에서 개발한 IT 인프라를 위한 표준 RESTful API

pm_type

이 옵션을 **redfish**로 설정합니다.

pm_user; pm_password

Redfish 사용자 이름 및 암호입니다.

pm_addr

Redfish 컨트롤러의 IP 주소입니다.

pm_system_id

시스템 리소스에 대한 표준 경로입니다. 이 경로에는 루트 서비스, 버전 및 시스템에 대한 경로/고유 ID가 포함되어야 합니다. 예: **/redfish/v1/Systems/CX34R87**

redfish_verify_ca

BMC(Baseboard Management Controller)의 Redfish 서비스에서 공인된 인증 기관(CA)이 서명한 올바른 TLS 인증서를 사용하도록 구성되지 않은 경우 Ironic의 Redfish 클라이언트가 BMC에 연결하지 못합니다. 오류를 표시하지 않으려면 **redfish_verify_ca** 옵션을 **false**로 설정합니다. 그러나 BMC 인증을 비활성화하면 BMC에 대한 액세스 보안이 손상될 수 있습니다.

B.2. DRAC(DELL REMOTE ACCESS CONTROLLER)

DRAC는 전원 관리 및 서버 모니터링을 포함하여 대역 외 원격 관리 기능을 제공하는 인터페이스입니다.

pm_type

이 옵션을 **idrac**로 설정합니다.

pm_user; pm_password

DRAC 사용자 이름 및 암호입니다.

pm_addr

DRAC 호스트의 IP 주소입니다.

B.3. ILO(INTEGRATED LIGHTS-OUT)

Hewlett-Packard의 iLO는 전원 관리 및 서버 모니터링을 포함하여 대역 외 원격 관리 기능을 제공하는 인터페이스입니다.

pm_type

이 옵션을 **ilo**로 설정합니다.

pm_user; pm_password

iLO 사용자 이름 및 암호입니다.

pm_addr

iLO 인터페이스의 IP 주소입니다.

- 이 드라이버를 활성화하려면 **undercloud.conf**의 **enabled_hardware_types** 옵션에 **ilo**를 추가하고 **openstack undercloud install**을 재실행합니다.
- director에는 iLo에 대한 추가 유틸리티 세트도 필요합니다. **python-proliantutils** 패키지를 설치하고 **openstack-ironic-conductor** 서비스를 다시 시작합니다.

```
$ sudo yum install python-proliantutils
$ sudo systemctl restart openstack-ironic-conductor.service
```

- 인트로스펙션에 성공하려면 HP 노드에 최소 ILO 펌웨어 버전 1.85(2015년 5월 13일)가 있어야 합니다. director는이 ILO 펌웨어 버전을 사용하는 노드에서 성공적으로 테스트되었습니다.
- 공유 iLO 포트 사용은 지원되지 않습니다.

B.4. CISCO UCS(UNIFIED COMPUTING SYSTEM)

참고

Cisco UCS는 더 이상 사용되지 않으며 RHOSP(Red Hat OpenStack Platform) 16.0에서 제거됩니다.

Cisco의 UCS는 컴퓨팅, 네트워크, 스토리지 액세스 및 가상화 리소스를 결합한 데이터 센터 플랫폼입니다. 이 드라이버는 UCS에 연결된 베어 메탈 시스템의 전원 관리에 중점을 둡니다.

pm_type
: 이 옵션을 **cisco-ucs-managed**로 설정합니다.

pm_user; pm_password
: UCS 사용자 이름 및 암호입니다.

pm_addr
: UCS 인터페이스의 IP 주소입니다.

pm_service_profile
: 사용할 UCS 서비스 프로필입니다. 일반적으로 **org-root/ls-[service_profile_name]** 형식을 사용합니다. 예를 들면 다음과 같습니다.

```
"pm_service_profile": "org-root/ls-Nova-1"
```

- 이 드라이버를 활성화하려면 **cisco-ucs-managed** 를 **undercloud.conf** 의 **enabled_hardware_types** 옵션에 추가하고 **openstack undercloud install** 을 재실행합니다.
- director에는 UCS에 대한 추가 유틸리티 세트도 필요합니다. **python-UcsSdk** 패키지를 설치하고 **openstack-ironic-conductor** 서비스를 다시 시작합니다.

```
$ sudo yum install python-UcsSdk
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.5. FUJITSU IRMC(INTEGRATED REMOTE MANAGEMENT CONTROLLER)

Fujitsu의 iRMC는 통합된 LAN 연결 및 확장된 기능이 있는 BMC(Baseboard Management Controller)입니다. 이 드라이버는 iRMC에 연결된 베어 메탈 시스템의 전원 관리에 중점을 둡니다.

중요

iRMC S4 이상이 필요합니다.

pm_type

이 옵션을 **irmc**로 설정합니다.

pm_user; pm_password

iRMC 인터페이스에 대한 사용자 이름 및 암호입니다.

pm_addr

iRMC 인터페이스의 IP 주소입니다.

pm_port(선택 사항)

iRMC 작업에 사용할 포트입니다. 기본값은 443입니다.

pm_auth_method(선택 사항)

iRMC 작업에 대한 인증 방법입니다. **basic** 또는 **digest**를 사용합니다. 기본값은 **basic**입니다.

pm_client_timeout(선택 사항)

iRMC 작업에 대한 타임아웃(초)입니다. 기본값은 60초입니다.

pm_sensor_method(선택 사항)

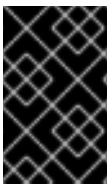
센서 데이터 검색 방법입니다. **ipmitool** 또는 **scci**를 사용합니다. 기본값은 **ipmitool**입니다.

- 이 드라이버를 활성화하려면 **irmc** 를 **undercloud.conf** 의 **enabled_hardware_types** 옵션에 추가하고 **openstack undercloud install** 을 재실행합니다.
- SCCI를 센서 방법으로 활성화한 경우 director에는 추가 유틸리티 세트도 필요합니다. **python-scciclient** 패키지를 설치하고 **openstack-ironic-conductor** 서비스를 다시 시작합니다.

```
$ yum install python-scciclient
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.6. VBMC(VIRTUAL BASEBOARD MANAGEMENT CONTROLLER)

director는 가상 머신을 KVM 호스트의 노드로 사용할 수 있습니다. 에뮬레이션된 IPMI 장치를 통해 전원 관리를 제어합니다. 이를 통해 6.1절. "오버클라우드에 노드 등록" 의 표준 IPMI 매개 변수를 사용할 수 있지만 가상 노드에는 사용할 수 있습니다.



중요

이 옵션은 베어 메탈 노드 대신 가상 머신을 사용합니다. 즉, 테스트 및 평가 목적으로만 사용할 수 있습니다. Red Hat OpenStack Platform 엔터프라이즈 환경에는 사용하지 않는 것이 좋습니다.

KVM 호스트 구성

1. KVM 호스트에서 OpenStack Platform 리포지토리를 활성화하고 **python2-virtualbmc** 패키지를 설치합니다.

```
$ sudo subscription-manager repos --enable=rhel-7-server-openstack-13-rpms
$ sudo yum install -y python2-virtualbmc
```

2. **vbmc** 명령을 사용하여 각 가상 머신에 대한 가상 BMC(Baseboard Management Controller)를 생성합니다. 예를 들어 **Node01** 및 **Node02** 라는 가상 머신에 대한 BMC를 생성하려면 각 BMC에 액세스할 포트를 정의하고 인증 세부 정보를 설정하려면 다음 명령을 입력합니다.

```
$ vbmc add Node01 --port 6230 --username admin --password PASSWORD
$ vbmc add Node02 --port 6231 --username admin --password PASSWORD
```

3. 호스트에서 해당 포트를 엽니다.

```
$ sudo firewall-cmd --zone=public \
--add-port=6230/udp \
--add-port=6231/udp
```

4. 변경 사항을 영구적으로 적용합니다.

```
$ sudo firewall-cmd --runtime-to-permanent
```

5. 변경 사항이 방화벽 설정에 적용되고 포트가 열려 있는지 확인합니다.

```
$ sudo firewall-cmd --list-all
```

참고

각 가상 머신에 다른 포트를 사용합니다. 1025 미만의 포트 번호에는 시스템에서 root 권한이 필요합니다.

6. 다음 명령을 사용하여 생성한 각 BMC를 시작합니다.

```
$ vbmc start Node01
$ vbmc start Node02
```

참고

KVM 호스트를 재부팅한 후 이 단계를 반복해야 합니다.

7. **ipmitool** 을 사용하여 노드를 관리할 수 있는지 확인하려면 원격 노드의 전원 상태를 표시합니다.

```
$ ipmitool -I lanplus -U admin -P PASSWORD -H 127.0.0.1 -p 6231 power status
Chassis Power is off
```

노드 등록

/home/stack/instackenv.json 노드 등록 파일에서 다음 매개 변수를 사용합니다.

pm_type

이 옵션을 **ipmi**로 설정합니다.

pm_user; pm_password

노드의 가상 BMC 장치에 대한 IPMI 사용자 이름 및 암호를 지정합니다.

pm_addr

노드가 포함된 KVM 호스트의 IP 주소를 지정합니다.

pm_port

KVM 호스트에서 특정 노드에 액세스할 포트를 지정합니다.

mac

노드에 있는 네트워크 인터페이스의 MAC 주소 목록을 지정합니다. 각 시스템의 프로비저닝 NIC에는 MAC 주소만 사용합니다.

예를 들면 다음과 같습니다.

```
{
  "nodes": [
    {
      "pm_type": "ipmi",
      "mac": [
        "aa:aa:aa:aa:aa:aa"
      ],
      "pm_user": "admin",
      "pm_password": "p455w0rd!",
      "pm_addr": "192.168.0.1",
      "pm_port": "6230",
      "name": "Node01"
    },
    {
      "pm_type": "ipmi",
      "mac": [
        "bb:bb:bb:bb:bb:bb"
      ],
      "pm_user": "admin",
      "pm_password": "p455w0rd!",
      "pm_addr": "192.168.0.1",
      "pm_port": "6231",
      "name": "Node02"
    }
  ]
}
```

기존 노드 마이그레이션

더 이상 사용되지 않는 **pxe_ssh** 드라이버를 사용하여 기존 노드를 새 가상 BMC 방법을 사용하도록 마이그레이션할 수 있습니다. 다음 명령은 **ipmi** 드라이버 및 해당 매개변수를 사용하도록 노드를 설정하는 예입니다.

```
openstack baremetal node set Node01 \
    --driver ipmi \
    --driver-info ipmi_address=192.168.0.1 \
    --driver-info ipmi_port=6230 \
    --driver-info ipmi_username="admin" \
    --driver-info ipmi_password="p455w0rd!"
```

B.7. RED HAT VIRTUALIZATION

이 드라이버는 RESTful API를 통해 Red Hat Virtualization에서 가상 머신을 제어합니다.

pm_type

이 옵션을 **staging-ovirt**로 설정합니다.

pm_user; pm_password

Red Hat Virtualization 환경에 대한 사용자 이름과 암호입니다. 사용자 이름에는 인증 제공업체 또한 포함되어 있습니다. 예: **admin@internal**

pm_addr

Red Hat Virtualization REST API의 IP 주소입니다.

pm_vm_name

제어할 가상 머신의 이름입니다.

mac

노드에 있는 네트워크 인터페이스의 MAC 주소 목록입니다. 각 시스템의 프로비저닝 NIC에는 MAC 주소만 사용합니다.

이 드라이버를 활성화하려면 다음 단계를 완료합니다.

1. **undercloud.conf** 파일의 **enabled_hardware_types** 옵션에 **staging-ovirt** 를 추가합니다.

```
enabled_hardware_types = ipmi,staging-ovirt
```

2. **python-ovirt-engine-sdk4.x86_64** 패키지를 설치합니다.

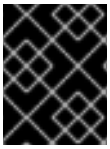
```
$ sudo yum install python-ovirt-engine-sdk4
```

3. **openstack undercloud install** 명령을 실행합니다.

```
$ openstack undercloud install
```

B.8. 페이크 드라이버

이 드라이버는 전원 관리 없이 베어 메탈 장치를 사용하는 방법을 제공합니다. 즉, director는 등록된 베어 메탈 장치를 제어하지 않으므로 인트로스펙션 및 배포 프로세스의 특정 시점에서 전원을 수동으로 제어해야 합니다.



중요

이 옵션은 테스트 및 평가 목적으로만 사용할 수 있습니다. Red Hat OpenStack Platform 엔터프라이즈 환경에는 사용하지 않는 것이 좋습니다.

pm_type

이 옵션을 **fake_pxe** 로 설정합니다.

- 이 드라이버는 전원 관리를 제어하지 않으므로 인증 세부 정보를 사용하지 않습니다.
- 이 드라이버를 활성화하려면 **fake_pxe** 를 **undercloud.conf** 의 **enabled_drivers** 옵션에 추가하고 **openstack undercloud install** 을 재실행합니다.
- **instackenv.json** 노드 인벤토리 파일에서 수동으로 관리하려는 노드의**pm_type** 을 **fake_pxe** 로 설정합니다.

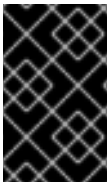
- 노드에서 인트로스펙션을 수행할 때 **openstack overcloud node introspect** 명령을 실행한 후 노드의 전원을 수동으로 켭니다.

- 오버클라우드 배포를 수행할 때 **ironic node-list** 명령을 사용하여 노드 상태를 확인합니다. 노드 상태가 **deploying** 에서 **deploy wait-callback** 으로 변경될 때까지 기다린 다음 노드의 전원을 수동으로 켭니다.

- 오버클라우드 프로비저닝 프로세스가 완료되면 노드를 다시 시작합니다. 프로비저닝이 완료되었는지 확인하려면 **ironic node-list** 명령을 사용하여 노드 상태를 확인하고, 노드 상태가 **active**로 변경될 때까지 기다린 다음 모든 오버클라우드 노드를 수동으로 재부팅합니다.

부록 C. 전체 디스크 이미지

기본 오버클라우드 이미지는 플랫 파티션 이미지입니다. 즉, 이미지 자체에 파티셔닝 정보 또는 부트로더가 포함되어 있지 않습니다. director는 부팅 시 별도의 커널 및 램디스크를 사용하고 오버클라우드 이미지를 디스크에 쓸 때 기본 파티션 레이아웃을 생성합니다. 하지만 파티션 레이아웃, 부트로더 및 강화된 보안 기능이 포함된 전체 디스크 이미지를 생성할 수 있습니다.



중요

다음 프로세스는 director의 이미지 구축 기능을 사용합니다. Red Hat은 이 섹션에 포함된 가이드라인을 사용하여 빌드된 이미지만 지원합니다. 이 사양과 다르게 빌드된 사용자 지정 이미지는 지원되지 않습니다.

보안 강화 이미지에는 보안이 중요한 기능인 Red Hat OpenStack Platform 배포에 필요한 추가 보안 조치가 포함되어 있습니다. 보안 이미지에 대한 권장 사항은 다음과 같습니다.

- **/tmp** 디렉터리가 별도의 볼륨이나 파티션에 마운트되어 있고**rw, nosuid, nodev, noexec, relatime** 플래그가 있습니다.
- **/var,/var/log, /var/log/audit** 디렉터리는 별도의 볼륨 또는 파티션에 마운트되어 있고**rw ,relatime** 플래그가 있습니다.
- **/home** 디렉터리는 별도의 파티션이나 볼륨에 마운트되어 있고**rw,nodev,relatime** 플래그가 있습니다.
- **GRUB_CMDLINE_LINUX** 설정을 다음과 같이 변경합니다.
  - 감사를 활성화하려면 **audit=1**을 추가하여 추가 커널 부팅 플래그를 포함합니다.
  - 부트 로더 구성을 사용하는 USB에 대한 커널 지원을 비활성화하려면 **nousb**를 추가합니다.
  - 비보안 부트 플래그를 삭제하려면 **crashkernel=auto**를 설정합니다.
- 비보안 모듈(**usb-storage,cramfs,freevxfs,jffs2,hfs** plus **,hfsplus,squashfs,udf,vfat**)을 블랙리스트로 지정하고 로드되지 않도록 합니다.
- 기본적으로 설치한 이미지에서 비보안 패키지(**kexec-tools** 및 **telnet**이 설치한 **kdump**)를 삭제합니다.
- 보안에 필요한 새 **screen** 패키지를 추가합니다.

보안 강화 이미지를 빌드하려면 다음이 필요합니다.

1. 기본 Red Hat Enterprise Linux 7 이미지 다운로드
2. 등록과 관련된 환경 변수 설정
3. 파티션 스키마 및 크기를 수정하여 이미지 사용자 지정
4. 이미지 생성
5. 배포에 업로드

다음 섹션에서는 이러한 작업을 수행하기 위한 절차를 자세히 설명합니다.

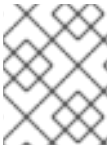
C.1. 기본 클라우드 이미지 다운로드

전체 디스크 이미지를 빌드하기 전에 기준으로 사용할 Red Hat Enterprise Linux의 기존 클라우드 이미지를 다운로드해야 합니다. Red Hat 고객 포털로 이동하여 다운로드할 KVM 게스트 이미지를 선택합니다. 예를 들어 최신 Red Hat Enterprise Linux의 KVM 게스트 이미지는 다음 페이지에서 이용할 수 있습니다.

- "Installers and Images for Red Hat Enterprise Linux Server"

C.2. 디스크 이미지 환경 변수

디스크 이미지 빌드 프로세스의 일환으로 director는 새 오버클라우드 이미지 패키지를 가져오기 위해 기본 이미지와 등록 세부 정보가 필요합니다. Linux 환경 변수를 사용하여 이러한 측면을 정의합니다.



참고

이미지 빌드 프로세스는 Red Hat 서브스크립션에서 이미지를 임시로 등록하고, 이미지 빌드 프로세스가 완료되면 시스템을 등록 취소합니다.

디스크 이미지를 빌드하려면 사용자의 환경과 요건에 맞는 Linux 환경 변수를 설정합니다.

DIB_LOCAL_IMAGE
: 기준으로 사용할 로컬 이미지를 설정합니다.

REG_ACTIVATION_KEY
: 등록 프로세스의 일부로 활성화 키를 대신 사용합니다.

REG_AUTO_ATTACH
: 가장 호환 가능한 서브스크립션을 자동으로 연결할지 여부를 정의합니다.

REG_BASE_URL
: 패키지를 가져올 콘텐츠 전달 서버의 기본 URL입니다. 기본 Customer Portal 서브스크립션 관리 프로세스는 **https://cdn.redhat.com**을 사용합니다. Red Hat Satellite 6 서버를 사용하는 경우 이 매개변수는 Satellite 서버의 기본 URL을 사용해야 합니다.

REG_ENVIRONMENT
: 조직 내의 환경에 등록합니다.

REG_METHOD
: 등록 방법을 설정합니다. Red Hat 고객 포털에 시스템을 등록하려면 **portal**을 사용합니다. Red Hat Satellite 6에 시스템을 등록하려면 **satellite**를 사용합니다.

REG_ORG
: 이미지를 등록할 조직입니다.

REG_POOL_ID
: 제품 서브스크립션 정보의 풀 ID입니다.

REG_PASSWORD
: 이미지를 등록하는 사용자 계정에 암호를 제공합니다.

REG_REPOS
: 리포지토리 이름 문자열은 쉼표(공백 없음)로 구분됩니다. 이 문자열의 각 리포지토리는 **subscription-manager**를 통해 활성화됩니다.
보안이 강화된 전체 디스크 이미지에 다음 리포지토리를 사용합니다.

  - **rhel-7-server-rpms**
  - **rhel-7-server-extras-rpms**

- **rhel-ha-for-rhel-7-server-rpms**
- **rhel-7-server-optional-rpms**
- **rhel-7-server-openstack-13-rpms**

REG_SAT_URL

오버클라우드 노드를 등록할 Satellite 서버의 기본 URL입니다. Satellite의 HTTPS URL 대신 HTTP URL을 이 매개변수에 사용합니다. 예를 들어 https://satellite.example.com 대신 http://satellite.example.com을 사용합니다.

REG_SERVER_URL

사용할 서브스크립션 서비스의 호스트 이름을 제공합니다. Red Hat Customer Portal에 대한 기본값은 **subscription.rhn.redhat.com**에 있습니다. Red Hat Satellite 6 서버를 사용하는 경우 이 매개 변수는 Satellite 서버의 호스트 이름을 사용해야 합니다.

REG_USER

이미지를 등록하는 계정에 사용자 이름을 제공합니다.

다음은 로컬 QCOW2 이미지를 Red Hat Customer Portal에 임시로 등록하기 위해 환경 변수 세트를 내보내는 명령의 예입니다.

```
$ export DIB_LOCAL_IMAGE=./rhel-server-7.5-x86_64-kvm.qcow2
$ export REG_METHOD=portal
$ export REG_USER="[your username]"
$ export REG_PASSWORD="[your password]"
$ export REG_REPOS="rhel-7-server-rpms \
    rhel-7-server-extras-rpms \
    rhel-ha-for-rhel-7-server-rpms \
    rhel-7-server-optional-rpms \
    rhel-7-server-openstack-13-rpms"
```

C.3. 디스크 레이아웃 사용자 지정

기본 보안 강화 이미지 크기는 20G이며 사전 정의된 파티션 크기를 사용합니다. 그러나 오버클라우드 컨테이너 이미지를 수용하려면 파티션 레이아웃을 일부 수정해야 합니다. 다음 섹션에서는 이미지 크기를 40G로 늘립니다. 필요에 따라 파티션 레이아웃과 디스크 크기를 추가로 수정할 수도 있습니다.

파티션 레이아웃과 디스크 크기를 수정하려면 다음 단계를 수행합니다.

- **DIB_BLOCK_DEVICE_CONFIG** 환경 변수를 사용하여 파티션 스키마를 수정합니다.
- **DIB_IMAGE_SIZE** 환경 변수를 업데이트하여 이미지 전체 크기를 수정합니다.

C.3.1. 파티션 스키마 수정

파티션 스키마를 수정하여 파티션 크기를 변경하거나, 새 파티션을 생성하거나, 기존 파티션을 삭제할 수 있습니다. 다음 환경 변수를 사용하여 새 파티션 스키마를 정의할 수 있습니다.

```
$ export DIB_BLOCK_DEVICE_CONFIG='<yaml_schema_with_partitions>'
```

다음 YAML 구조는 오버클라우드 컨테이너 이미지를 가져올 충분한 공간을 수용하기 위해 수정된 논리 볼륨 파티션 레이아웃을 나타냅니다.

```
export DIB_BLOCK_DEVICE_CONFIG='''
- local_loop:
    name: image0
- partitioning:
    base: image0
    label: mbr
    partitions:
      - name: root
        flags: [ boot,primary ]
        size: 40G
- lvm:
    name: lvm
    base: [ root ]
    pvs:
        - name: pv
          base: root
          options: [ "--force" ]
    vgs:
        - name: vg
          base: [ "pv" ]
          options: [ "--force" ]
    lvs:
        - name: lv_root
          base: vg
          extents: 23%VG
        - name: lv_tmp
          base: vg
          extents: 4%VG
        - name: lv_var
          base: vg
          extents: 45%VG
        - name: lv_log
          base: vg
          extents: 23%VG
        - name: lv_audit
          base: vg
          extents: 4%VG
        - name: lv_home
          base: vg
          extents: 1%VG
- mkfs:
    name: fs_root
    base: lv_root
    type: xfs
    label: "img-rootfs"
    mount:
        mount_point: /
        fstab:
            options: "rw,relatime"
            fsck-passno: 1
- mkfs:
    name: fs_tmp
    base: lv_tmp
    type: xfs
    mount:
        mount_point: /tmp
```

```
        fstab:
          options: "rw,nosuid,nodev,noexec,relatime"
          fsck-passno: 2
- mkfs:
    name: fs_var
    base: lv_var
    type: xfs
    mount:
      mount_point: /var
      fstab:
        options: "rw,relatime"
        fsck-passno: 2
- mkfs:
    name: fs_log
    base: lv_log
    type: xfs
    mount:
      mount_point: /var/log
      fstab:
        options: "rw,relatime"
        fsck-passno: 3
- mkfs:
    name: fs_audit
    base: lv_audit
    type: xfs
    mount:
      mount_point: /var/log/audit
      fstab:
        options: "rw,relatime"
        fsck-passno: 4
- mkfs:
    name: fs_home
    base: lv_home
    type: xfs
    mount:
      mount_point: /home
      fstab:
        options: "rw,nodev,relatime"
        fsck-passno: 2
'"
```

이 샘플 YAML 콘텐츠를 이미지 파티션 스키마의 기준으로 사용합니다. 필요에 따라 파티션 크기와 레이아웃을 수정합니다.

참고

배포 후에는 파티션 크기를 조정할 수 없으므로 이미지에 대해 올바른 파티션 크기를 정의합니다.

C.3.2. 이미지 크기 수정

수정된 파티션 스키마의 총합이 기본 디스크 크기(20G)를 초과할 수도 있습니다. 이 경우에는 이미지 크기를 수정해야 합니다. 이미지 크기를 수정하려면 이미지를 생성하는 데 사용되는 구성 파일을 편집합니다.

/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images.yaml 사본을 생성합니다.

```
# cp /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images.yaml \
/home/stack/overcloud-hardened-images-custom.yaml
```

참고

UEFI 전체 디스크 이미지의 경우 /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-uefi.yaml 을 사용합니다.

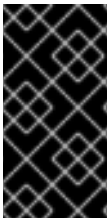
구성 파일에서 **DIB_IMAGE_SIZE** 를 편집하여 필요에 따라 값을 조정합니다.

```
...

environment:
DIB_PYTHON_VERSION: '2'
DIB_MODPROBE_BLACKLIST: 'usb-storage cramfs freevxfs jffs2 hfs hfsplus squashfs udf vfat
bluetooth'
DIB_BOOTLOADER_DEFAULT_CMDLINE: 'nofb nomodeset vga=normal console=tty0
console=ttyS0,115200 audit=1 nousb'
DIB_IMAGE_SIZE: '40' (1)
COMPRESS_IMAGE: '1'
```

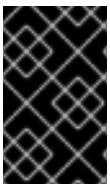
1 이 값을 새로운 디스크 전체 크기에 맞게 조정합니다.

이 파일을 저장합니다.



중요

director가 오버클라우드를 배포할 때 오버클라우드 이미지의 RAW 버전을 생성합니다. 따라서 언더클라우드에 RAW 이미지를 수용하는 데 필요한 여유 공간이 있어야 합니다. 예를 들어 보안 강화 이미지 크기를 40G로 늘리는 경우 언더클라우드의 하드 디스크에 40G의 사용 가능한 공간이 있어야 합니다.



중요

결국 director가 물리 디스크에 이미지를 쓸 때 디스크 끝에 64MB의 구성 드라이브 주 파티션을 생성합니다. 전체 디스크 이미지를 생성할 때 이 추가 파티션을 수용할 수 있도록 물리 디스크 크기보다 작은지 확인합니다.

C.4. 보안 강화 전체 디스크 이미지 생성

환경 변수를 설정하고 이미지를 사용자 지정한 후 openstack overcloud image build 명령을 사용하여 이미지를 생성합니다.

```
# openstack overcloud image build \
--image-name overcloud-hardened-full \
--config-file /home/stack/overcloud-hardened-images-custom.yaml \
--config-file /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-
rhel7.yaml
```

/home/stack/overcloud-hardened-images-custom.yaml 사용자 지정 구성 파일에는C.3.2절. "이미지 크기 수정" 의 새 디스크 크기가 포함되어 있습니다. 다른 사용자 지정 디스크 크기를 사용하지않는 경우 기존 **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images.yaml** 파일을 대신 사용하십시오.

UEFI 전체 디스크 이미지의 경우 **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-uefi-rhel7.yaml** 구성 파일을 사용합니다.

생성한 **overcloud-hardened-full.qcow2** 이미지에는 필요한 모든 보안 기능이 포함되어 있습니다.

C.5. 보안 강화 전체 디스크 이미지 업로드

이미지를 OpenStack Image(glance) 서비스에 업로드하고 Red Hat OpenStack Platform director에서 이를 사용하기 시작합니다. 보안 강화 이미지를 업로드하려면 다음 단계를 실행합니다.

1. 새로 생성된 이미지의 이름을 변경하고 이미지 디렉터리로 이동합니다.

```
# mv overcloud-hardened-full.qcow2 ~/images/overcloud-full.qcow2
```

2. 이전 오버클라우드 이미지를 모두 삭제합니다.

```
# openstack image delete overcloud-full
# openstack image delete overcloud-full-initrd
# openstack image delete overcloud-full-vmlinuz
```

3. 새 오버클라우드 이미지를 업로드합니다.

```
# openstack overcloud image upload --image-path /home/stack/images --whole-disk
```

기존 이미지를 보안 강화 이미지로 교체하려면 **--update-existing** 플래그를 사용합니다. 그러면 원래 **overcloud-full** 이미지를 사용자가 생성한 새로운 보안 강화 이미지로 덮어씁니다.

부록 D. 대체 부팅 모드

노드의 기본 부팅 모드는 iPXE를 통한 BIOS입니다. 다음 섹션에서는 director에서 노드를 프로비저닝하고 검사할 때 사용할 몇 가지 대체 부팅 모드를 간략하게 설명합니다.

D.1. 표준 PXE

iPXE 부팅 프로세스는 HTTP를 사용하여 인트로스펙션 및 배포 이미지를 부팅합니다. 이전 시스템은 TFTP를 통해 부팅하는 표준 PXE 부팅만 지원할 수 있습니다.

iPXE에서 PXE로 변경하려면 director 호스트에서 **undercloud.conf** 파일을 편집하고 **ipxe_enabled** 를 **False** 로 설정합니다.

```
ipxe_enabled = False
```

이 파일을 저장하고 언더클라우드 설치를 실행합니다.

```
$ openstack undercloud install
```

이 프로세스에 대한 자세한 내용은 "Red Hat OpenStack Platform director에서 PXE로 iPXE 변경"문서를 참조하십시오.

D.2. UEFI 부팅 모드

기본 부팅 모드는 기존 BIOS 모드입니다. 최신 시스템에는 기존 BIOS 모드 대신 UEFI 부팅 모드가 필요할 수 있습니다. 이 경우 **undercloud.conf** 파일에 다음을 설정합니다.

```
ipxe_enabled = True
inspection_enable_uefi = True
```

이 파일을 저장하고 언더클라우드 설치를 실행합니다.

```
$ openstack undercloud install
```

등록된 각 노드의 부팅 모드를 **uefi**로 설정합니다. 예를 들어 **capabilities** 속성에서 기존 **boot_mode** 매개변수를 추가하거나 교체하려면 다음을 수행합니다.

```
$ NODE=<NODE NAME OR ID> ; openstack baremetal node set --property
capabilities="boot_mode:uefi,$(openstack baremetal node show $NODE -f json -c properties | jq -r
.properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

참고

이 명령을 사용하여 **profile** 및 **boot_option** 기능이 유지되고 있는지 확인합니다.

또한 각 플레이버의 부팅 모드를 **uefi** 로 설정합니다. 예를 들면 다음과 같습니다.

```
$ openstack flavor set --property capabilities:boot_mode='uefi' control
```

부록 E. 프로파일 자동 태그

인트로스펙션 프로세스는 일련의 벤치마크 테스트를 수행합니다. director는 이러한 테스트의 데이터를 저장합니다. 이 데이터를 사용하는 정책 세트를 다양한 방식으로 생성할 수 있습니다. 예를 들면 다음과 같습니다.

- 정책을 통해 성능이 떨어지거나 불안정한 노드를 오버클라우드에서 사용하지 않도록 식별하고 분리할 수 있습니다.
- 정책을 통해 노드를 특정 프로필에 자동으로 태그할지 여부를 정의할 수 있습니다.

E.1. 정책 파일 구문

정책 파일은 규칙 세트가 포함된 JSON 포멧을 사용합니다. 각 규칙은 *description*, *condition* 및 *action*을 정의합니다.

설명

일반 텍스트로 작성된 규칙 설명입니다.

예제:

```
"description": "A new rule for my node tagging policy"
```

조건

조건은 다음 키-값 패턴을 사용하여 평가를 정의합니다.

field
: 평가할 필드를 정의합니다. 필드 유형의 경우 참조 E.4절. “프로파일 자동 태그 속성”

op
: 평가에 사용할 작업을 정의합니다. 여기에는 다음이 포함됩니다.

  - **eq** - 같음
  - **ne** - 같지 않음
  - **lt** - 보다 작음
  - **gt** - 보다 큼
  - **le** - 작거나 같음
  - **ge** - 크거나 같음
  - **in-net** - IP 주소가 지정된 네트워크에 있는지 확인
  - **matches** - 지정된 정규 표현과 완전히 일치해야 함
  - **contains** - 지정된 정규 표현식을 포함하는 값이 필요함
  - **is-empty** - 빌드가 비어 있는지 확인

invert
: 평가 결과를 반전할지 여부를 정의하는 부울 값입니다.

multiple

여러 결과가 있는 경우 사용할 평가를 정의합니다. 여기에는 다음이 포함됩니다.

- **any** - 임의 결과가 일치해야 함
- **all** - 모든 결과가 일치해야 함
- **first** - 첫 번째 결과가 일치해야 함

value

평가의 값을 정의합니다. 필드 및 작업 결과가 값이면 조건이 true 결과를 반환합니다. 그렇지 않으면 조건이 false를 반환합니다.

예제:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

작업

조건이 true로 반환되는 경우 작업이 수행됩니다. **action** 값에 따라 **action** 키와 추가 키를 사용합니다.

- **fail** - 인트로스펙션이 실패합니다. 실패 메시지에 대한 **message** 매개변수가 필요합니다.
- **set-attribute** - Ironic 노드의 특성을 설정합니다. Ironic 특성의 경로(예: **/driver_info/ipmi_address**)인 **path** 필드와 설정할 **value**가 필요합니다.
- **set-capability** - Ironic 노드의 기능을 설정합니다. 따라서 새 기능의이름과 값 인 name 및 value 필드가 필요합니다. 이 동일한 기능에 대한 기존 값은 교체됩니다. 예를 들면 노드 프로필을 정의하는 데 이 값을 사용합니다.
- **extend-attribute** - **set-attribute**와 동일하지만, 기존 값을 목록으로 처리하고 값을 여기에 추가합니다. 선택 사항인 **unique** 매개변수를 True로 설정하면, 지정된 값이 이미 목록에 있을 경우 아무것도 추가되지 않습니다.

예제:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

E.2. 정책 파일 예

다음은 적용할 인트로스펙션 규칙이 포함된 예제 JSON 파일(**rules.json**)입니다.

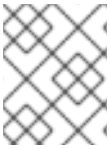
```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
        "op": "ge",
        "field": "local_gb",
        "value": 40
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "compute_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
```

```
        "name": "control_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "profile",
        "value": null
      }
    ]
  }
]
```

이 예제는 다음 세 가지 규칙으로 구성되어 있습니다.

- 메모리가 4096MiB 미만인 경우 인트로스펙션이 실패합니다. 이러한 규칙을 적용하여 클라우드의 일부가 되지 않아야 하는 노드를 제외하는 데 적용할 수 있습니다.
- 하드 드라이브 크기가 1TiB 이상인 노드에는 무조건 swift-storage 프로필이 할당됩니다.
- 하드 드라이브 크기가 40GiB보다 크고 1TiB 미만인 노드는 컴퓨팅 또는 컨트롤러 노드일 수 있습니다. 나중에 **openstack overcloud profiles match** 명령을 통해 최종 선택을 할 수 있도록 두 가지 기능(**compute_profile** 및 **control_profile**)을 할당합니다. 이를 위해 기존 프로필 기능을 제거하고, 그렇지 않으면 우선 순위를 갖습니다.

다른 노드는 변경되지 않습니다.



참고

인트로스펙션 규칙을 사용하여 **profile** 기능을 할당하면 기존 값이 항상 재정의됩니다. 그러나 기존 프로필 기능이 있는 노드에 대해서는 **[PROFILE]_profile** 기능이 무시됩니다.

E.3. 정책 파일 가져오기

다음 명령을 사용하여 정책 파일을 director로 가져옵니다.

```
$ openstack baremetal introspection rule import rules.json
```

그런 다음 인트로스펙션 프로세스를 실행합니다.

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

인트로스펙션이 완료되면 노드 및 할당된 해당 프로필을 확인합니다.

```
$ openstack overcloud profiles list
```

인트로스펙션 규칙에 오류가 있는 경우 모두 삭제할 수 있습니다.

```
$ openstack baremetal introspection rule purge
```

E.4. 프로파일 자동 태그 속성

자동 프로필 태그 지정은 각 조건의 **field** 속성에 대해 다음 노드 속성을 평가합니다.

속성	설명
memory_mb	노드의 메모리 크기(MB)입니다.
cpus	노드의 CPU의 총 코어 수입니다.
cpu_arch	노드 CPU의 아키텍처입니다.
local_gb	노드의 root 디스크 총 스토리지 공간입니다. 노드의 root 디스크 설정에 대한 자세한 내용은 6.6절. "root 디스크 정의" 을 참조하십시오.

부록 F. 보안 강화

다음 섹션에서는 언더클라우드의 보안을 강화하기 위한 몇 가지 제안 사항을 제공합니다.

F.1. HAPROXY에 대한 SSL/TLS 암호화 방식 및 규칙 변경

언더클라우드에서 SSL/TLS를 활성화한 경우(4.9절. "director 설정 매개변수"참조) HAProxy 구성에 사용되는 SSL/TLS 암호화 방식 및 규칙을 강화할 수 있습니다. 이를 통해 POODLE 취약점과 같은 SSL/TLS 취약점을 방지할 수 있습니다.

hieradata_override 언더클라우드 설정 옵션을 사용하여 다음 hieradata를 설정합니다.

tripleo::haproxy::ssl_cipher_suite

HAProxy에서 사용할 암호화 방식 세트입니다.

tripleo::haproxy::ssl_options

HAProxy에서 사용할 SSL/TLS 규칙입니다.

예를 들면 다음 암호화 방식 및 규칙을 사용할 수 있습니다.

- 암호화 방식: **ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-AES128-AES256-AES256-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-AES256-AES256-AES256-AES256-AES256-RSA-AES256-RSA-AES256-AES256-AES256-AES256-AES256-RSA-AES128-RSA-AES128-ECDSA-AES128-RSA-AES128-AES128-AHA256:ECDHE-RSA-RSA-AES128-AES128-AES128-AES128-GCM-AHA256:ECDHE-RSA-RSA-CHACHA20-AHA384:ECDHE-RSA-RSA-RSA-RSA-ECDSA-GCM-ECDSA-ECDSA-ECDSA-ECDSA-ECDSA-GCM-GCM-GCM-GCM-GCM-GCM:ECDHE- DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-AES256-AES128-AES128-SHA256:ECDHE-RSA-AES128-AES128-AES128-AES128-AES256-SHA384:ECDHE-RSA-AES256:ECDHE-RSA-AES256:ECDHE-RSA-AES128-AES128-AHA384:ECDHE-RSA-AES128-AES128-AES256:ECDHE-RSA-AES256:ECDHE-RSA-AES256:ECDHE-RSA-AES256:ECDHE-RSA-AES256-AES128-AES256:ECDHE-RSA-RSA-AES256-RSA-AES256:ECDHE-RSA-AES128-AES128-AHA384:ECDHE-RSA-AES128-AES128-AES128-AES128-AES256-AES256-AES256-AES256-AES256-AES128-AES256:ECDHE-RSA-RSA-GCM-SHA384:DHE-RSA-RSA-GCM-RSA-RSA-RSA-ECDHE-ECDSA-GCM-GCM -AES256-SHA384:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES256-AES128-SHA256:DHE-RSA-AES128-AES128-SHA:DHE-RSA-AES256-AES256-SHA256:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA256:DHE-RSA-AES256:DHE-RSA-AES256:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-AES128-SHA256:DHE-RSA-AES256-AES256-AES256-SHA:DHE-RSA-AES256-AES256-AES256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA: ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA:AES128-SHA:AES128-SHA:AES128-SHA256:AES128-SHA:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA256:AES128-SHA:AES128-SHA:AES128-SHA:AES128-GCM-SHA256:AES128-SHA256:AES128-SHA:AES128-SHA256:AES128-SHA256:AES128-SHA-SHA:AES128-SHA-SHA:AES128-GCM-SHA256:AES128-GCM-GCM:AES128-SHA-SHA:AES128-SHA-SHA:AES128-GCM-GCM:AES128-GCM-**
- 규칙: **no-sslv3 no-tls-tickets**

다음 콘텐츠로 hieradata 덮어쓰기 파일(**haproxy-hiera-overrides.yaml**)을 생성합니다.

-

```
tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-
CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-
SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-
CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

참고

암호화 방식 컬렉션은 연속된 한 행으로 되어 있습니다.

openstack undercloud install을 실행하기 전에 생성한 hieradata 재정의 파일을 사용하도록 **undercloud.conf** 파일의 **hieradata_override** 매개변수를 설정합니다.

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```

부록 G. RED HAT OPENSTACK PLATFORM FOR POWER

Red Hat OpenStack Platform을 새로 설치하는 경우 이제 POWER(ppc64le) 하드웨어에 오버클라우드 컴퓨팅 노드를 배포할 수 있습니다. 컴퓨팅 노드 클러스터의 경우 동일한 아키텍처를 사용하거나 x86_64 및 ppc64le 시스템을 혼합하여 사용하도록 선택할 수 있습니다. 언더클라우드, 컨트롤러 노드, Ceph Storage 노드 및 기타 모든 시스템은 x86_64 하드웨어에서만 지원됩니다. 각 시스템의 설치 세부 정보는 이 가이드의 이전 섹션에서 설명합니다.

G.1. CEPH STORAGE

다중 아키텍처 클라우드에서 외부 Ceph에 대한 액세스 권한을 설정하는 경우 클라이언트 키 및 기타 Ceph 특정 매개변수와 함께 **CephAnsiblePlaybook** 매개변수를 **/usr/share/ceph-ansible/site.yml.sample**로 설정합니다.

예를 들면 다음과 같습니다.

```
parameter_defaults:
  CephAnsiblePlaybook: /usr/share/ceph-ansible/site.yml.sample
  CephClientKey: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
  CephClusterFSID: 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
  CephExternalMonHost: 172.16.1.7, 172.16.1.8
```

G.2. 구성 가능 서비스

일반적으로 컨트롤러 노드의 일부인 다음 서비스는 사용자 지정 역할에 기술 프리뷰로 제공되므로 Red Hat에서 완전히 지원하지 않습니다.

- **Cinder**
- **Glance**
- **Keystone**
- **Neutron**
- **Swift**

자세한 내용은 composable services and custom roles문서를 참조하십시오. 다음은 나열된 서비스를 컨트롤러 노드에서 전용 ppc64le 노드로 이동하는 한 가지 방법입니다.

```
(undercloud) [stack@director ~]$ rsync -a /usr/share/openstack-tripleo-heat-templates/. ~/templates
(undercloud) [stack@director ~]$ cd ~/templates/roles
(undercloud) [stack@director roles]$ cat <<EO_TEMPLATE >ControllerPPC64LE.yaml
###############################################################################
# Role: ControllerPPC64LE                                                     #
###############################################################################
- name: ControllerPPC64LE
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
```

```
networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
# For systems with both IPv4 and IPv6, you may specify a gateway network for
# each, such as ['ControlPlane', 'External']
default_route_networks: ['External']
HostnameFormatDefault: '%stackname%-controllerppc64le-%index%'
ImageDefault: ppc64le-overcloud-full
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::CinderApi
  - OS::TripleO::Services::CinderBackendDellPs
  - OS::TripleO::Services::CinderBackendDellSc
  - OS::TripleO::Services::CinderBackendDellEMCUnity
  - OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
  - OS::TripleO::Services::CinderBackendDellEMCVNX
  - OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI
  - OS::TripleO::Services::CinderBackendNetApp
  - OS::TripleO::Services::CinderBackendScaleIO
  - OS::TripleO::Services::CinderBackendVRTSHyperScale
  - OS::TripleO::Services::CinderBackup
  - OS::TripleO::Services::CinderHPELeftHandISCSI
  - OS::TripleO::Services::CinderScheduler
  - OS::TripleO::Services::CinderVolume
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::Fluentd
  - OS::TripleO::Services::GlanceApi
  - OS::TripleO::Services::GlanceRegistry
  - OS::TripleO::Services::Ipsec
  - OS::TripleO::Services::Iscsid
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::Keystone
  - OS::TripleO::Services::LoginDefs
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::NeutronApi
  - OS::TripleO::Services::NeutronBgpVpnApi
  - OS::TripleO::Services::NeutronSfcApi
  - OS::TripleO::Services::NeutronCorePlugin
  - OS::TripleO::Services::NeutronDhcpAgent
  - OS::TripleO::Services::NeutronL2gwAgent
  - OS::TripleO::Services::NeutronL2gwApi
  - OS::TripleO::Services::NeutronL3Agent
  - OS::TripleO::Services::NeutronLbaasv2Agent
  - OS::TripleO::Services::NeutronLbaasv2Api
  - OS::TripleO::Services::NeutronLinuxbridgeAgent
  - OS::TripleO::Services::NeutronMetadataAgent
  - OS::TripleO::Services::NeutronML2FujitsuCfab
```

```
    - OS::TripleO::Services::NeutronML2FujitsuFossw
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftDispersion
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
EO_TEMPLATE
(undercloud) [stack@director roles]$ sed -i~ -e '/OS::TripleO::Services::\
(Cinder\|Glance\|Swift\|Keystone\|Neutron\)/d' Controller.yaml
(undercloud) [stack@director roles]$ cd ../
(undercloud) [stack@director templates]$ openstack overcloud roles generate \
   --roles-path roles -o roles_data.yaml \
   Controller Compute ComputePPC64LE ControllerPPC64LE BlockStorage ObjectStorage
CephStorage
```