Red Hat OpenStack Platform 16.2

Director 설치 및 사용

Red Hat OpenStack Platform director를 사용하여 OpenStack 클라우드 환경 생성

Last Updated: 2024-04-04

Red Hat OpenStack Platform 16.2 Director 설치 및 사용

Red Hat OpenStack Platform director를 사용하여 OpenStack 클라우드 환경 생성

OpenStack Team
rhos-docs@redhat.com

법적 공지

초록

Red Hat OpenStack Platform director를 사용하여 엔터프라이즈 환경에 Red Hat OpenStack Platform 16을 설치하십시오. 여기에는 director 설치, 환경 계획, director를 사용한 OpenStack 환경 구축 작업이 포함됩니다.

차례

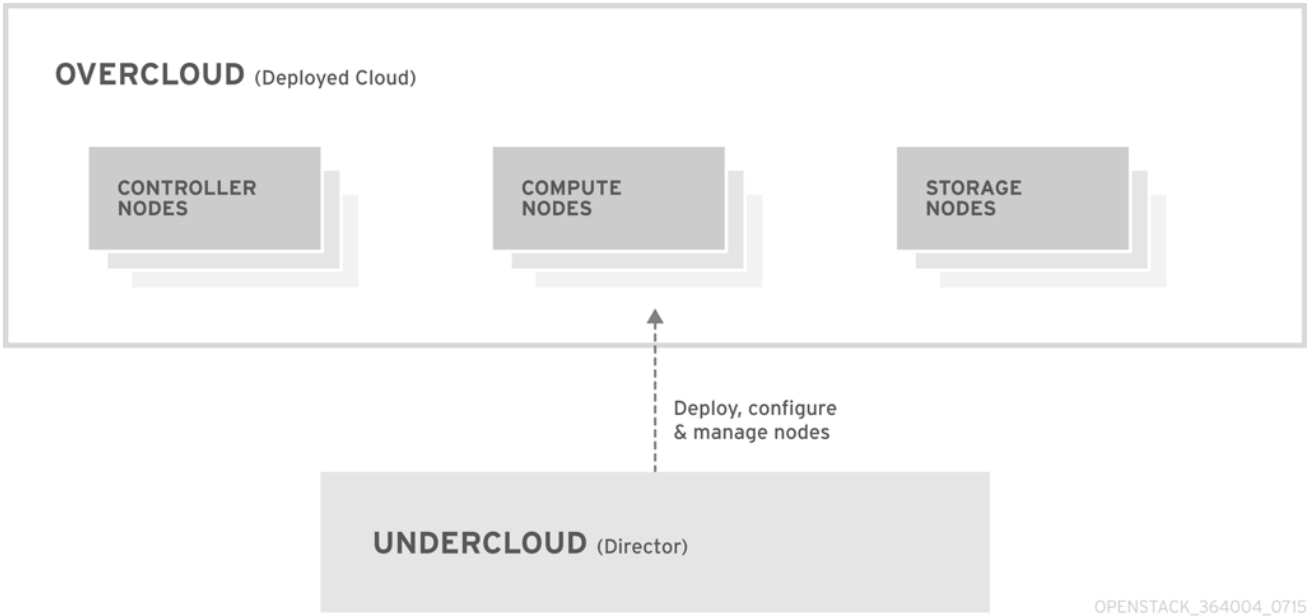
보다 포괄적 수용을 위한 오픈 소스 용어 교체

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

1장. DIRECTOR 소개

RHOSP(Red Hat OpenStack Platform) director는 완전한 RHOSP 환경을 설치하고 관리하기 위한 툴셋입니다. director는 주로 OpenStack 프로젝트 TripleO를 기반으로 합니다. director를 사용하여 모든 기능을 갖춘 가볍고 견고한 RHOSP 환경을 설치하고, OpenStack 노드로 사용할 베어 메탈 시스템을 프로비저닝 및 제어할 수 있습니다.

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 이해

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 계획

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 각 노드에 OpenStack을 설치하는 데 각 노드의 대역 외 관리 인터페이스(일반적으로 IPMI(Intelligent Platform Management Interface))를 사용합니다. 이 기능을 사용하여 베어 메탈 시스템을 OpenStack 노드로 프로비저닝할 수 있습니다. 전원 관리 드라이버의 전체 목록은 30장. *전원 관리 드라이버*를 참조하십시오.

오케스트레이션

언더클라우드에는 환경 계획 역할을 하는 YAML 템플릿 세트가 포함되어 있습니다. 언더클라우드는 이러한 환경 계획을 가져와서 해당 지침에 따라 원하는 OpenStack 환경을 생성합니다. 이 계획에는 환경 생성 프로세스 중 특정 시점에 사용자 지정을 통합하는 데 사용할 수 있는 후크도 포함되어 있습니다.

언더클라우드 구성 요소

언더클라우드는 OpenStack 구성 요소를 기본 툴셋으로 사용합니다. 각 구성 요소는 언더클라우드의 개별 컨테이너 내에서 작동합니다.

- OpenStack Identity(keystone) - director 구성 요소에 대한 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) - director가 베어 메탈 머신에 기록한 이미지를 저장합니다.
- OpenStack Orchestration(heat) 및 Puppet - director가 오버클라우드 이미지를 디스크에 기록한 후 노드의 오케스트레이션 및 구성을 제공합니다.
- 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 Service 배포 계획

1.2. 오버클라우드 이해

오버클라우드는 언더클라우드를 사용하여 구축된 RHOSP(Red Hat OpenStack Platform) 환경입니다. 오버클라우드는 구축하려는 OpenStack Platform 환경에 따라 각기 다른 역할이 정의된 여러 노드로 구성됩니다. 언더클라우드에는 다음과 같은 오버클라우드 노드 역할이 기본적으로 포함되어 있습니다.

컨트롤러

컨트롤러 노드는 OpenStack 환경 관리, 네트워킹 및 고가용성 기능을 제공합니다. 이상적인 OpenStack 환경은 고가용성 클러스터에 3개의 컨트롤러 노드를 함께 사용하는 것입니다.
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 Shared File Systems(manila)
- OpenStack Bare Metal(ironic)
- OpenStack Load Balancing-as-a-Service(octavia)
- OpenStack Key Manager(barbican)
- MariaDB
- Open vSwitch
- 고가용성 서비스를 위한 Pacemaker 및 Galera

컴퓨팅

컴퓨팅 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

스토리지 노드는 OpenStack 환경에 스토리지를 제공합니다. 다음 목록에서 RHOSP의 다양한 스토리지 노드 유형에 대해 설명합니다.

- Ceph Storage 노드 - 스토리지 클러스터를 만드는 데 사용됩니다. 각 노드에는 Ceph OSD(Object Storage Daemon)가 포함됩니다. 또한 director는 Ceph Storage 노드를 환경의 일부로 배포하는 경우 컨트롤러 노드에 Ceph Monitor를 설치합니다.
- Block Storage(cinder) - 고가용성 컨트롤러 노드를 위한 외부 블록 스토리지로 사용됩니다. 이 노드에는 다음 구성 요소가 포함됩니다.
  - OpenStack Block Storage(cinder) 볼륨
  - OpenStack Telemetry 에이전트
  - Open vSwitch
- Object Storage(swift) - 이러한 노드는 Openstack Swift에 외부 스토리지 계층을 제공합니다. 컨트롤러 노드는 Swift 프록시를 통해 오브젝트 스토리지 노드에 액세스합니다. 오브젝트 스토리지 노드에는 다음 구성 요소가 포함됩니다.
  - OpenStack Object Storage(swift) 스토리지
  - OpenStack Telemetry 에이전트
  - Open vSwitch

1.3. RED HAT OPENSTACK PLATFORM에서 고가용성 이해

RHOSP(Red Hat OpenStack Platform) director는 컨트롤러 노드 클러스터를 사용하여 OpenStack Platform 환경에 고가용성 서비스를 제공합니다. 각 서비스에 대해 director는 모든 컨트롤러 노드에 동일한 구성 요소를 설치하고, 이를 단일 서비스로 관리합니다. 이 유형의 클러스터 구성은 단일 컨트롤러 노드에서 작동 오류가 발생할 경우 폴백 메커니즘을 제공합니다. 이 기능을 통해 OpenStack 사용자는 일정 수준의 연속 작업을 수행할 수 있습니다.

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 - 클러스터에서 RHOSP 데이터베이스를 복제합니다.
- Memcached - 데이터베이스 캐싱을 제공합니다.

참고

- 버전 13 이상에서는 director를 사용하여 Compute 인스턴스의 고가용성(인스턴스 HA)을 배포할 수 있습니다. 인스턴스 HA를 사용하면 컴퓨팅 노드에 오류가 발생할 경우 컴퓨팅 노드의 인스턴스 비우기를 자동화할 수 있습니다.

1.4. RED HAT OPENSTACK PLATFORM에서 컨테이너화 이해

언더클라우드 및 오버클라우드의 각 OpenStack Platform 서비스는 각 노드의 개별 Linux 컨테이너 내에서 실행됩니다. 이러한 컨테이너화를 통해 서비스를 분리하고 환경을 유지 관리하며 RHOSP(Red Hat OpenStack Platform)를 업그레이드할 수 있습니다.

이제 Red Hat Enterprise Linux 8.4 운영 체제에 Red Hat OpenStack Platform 16.2를 설치할 수 있습니다. Red Hat Enterprise Linux 8.4에는 더 이상 Docker가 포함되지 않으며 Docker 에코시스템을 대체할 새로운 툴셋을 제공합니다. 따라서 OpenStack Platform 16.2에서는 Docker가 OpenStack Platform 배포 및 업그레이드를 위한 새로운 툴로 대체됩니다.

Podman

Podman(Pod Manager)은 컨테이너 관리 툴로, Docker Swarm과 관련된 명령을 제외한 거의 모든 Docker CLI 명령을 구현합니다. Podman은 포드, 컨테이너, 컨테이너 이미지를 관리합니다. Podman과 Docker의 주요 차이점 중 하나는 Podman의 경우 백그라운드에서 데몬을 실행하지 않고도 리소스를 관리할 수 있다는 것입니다.
Podman에 관한 자세한 내용은 Podman 웹 사이트를 참조하십시오.

Buildah

Buildah는 Podman과 함께 사용되는 OCI(Open Containers Initiative) 이미지를 빌드합니다. Buildah 명령은 Dockerfile의 콘텐츠를 복제합니다. 또한 Buildah는 컨테이너 이미지 빌드용으로 간단한 **coreutils** 인터페이스를 제공하므로 컨테이너를 빌드하는 데 Dockerfile이 필요하지 않습니다. 또한 Buildah는 다른 스크립팅 언어를 사용하여 데몬 없이도 컨테이너 이미지를 빌드합니다.
Buildah에 관한 자세한 내용은 Buildah 웹 사이트를 참조하십시오.

Skopeo

Skopeo를 통해 운영자는 원격 컨테이너 이미지를 검사하고 director가 이미지를 가져올 때 데이터를 수집할 수 있습니다. 추가 기능으로 레지스트리의 컨테이너 이미지를 다른 레지스트리로 복사하거나 레지스트리에서 이미지를 삭제할 수도 있습니다.

Red Hat에서는 다음과 같은 오버클라우드용 컨테이너 이미지 관리 방법을 지원합니다.

- Red Hat Container Catalog에서 언더클라우드의 **image-serve** 레지스트리로 컨테이너 이미지를 가져온 다음 **image-service** 레지스트리에서 이미지를 가져옵니다. 먼저 언더클라우드로 이미지를 가져오면 여러 오버클라우드 노드에서 동시에 외부 연결을 통해 컨테이너 이미지를 가져오는 일을 방지할 수 있습니다.

- Satellite 6 서버에서 컨테이너 이미지를 가져옵니다. 네트워크 트래픽은 내부 트래픽이므로 Satellite에서 직접 이러한 이미지를 가져올 수 있습니다.

이 가이드에서는 컨테이너 이미지 레지스트리 세부 정보를 구성하고 기본적인 컨테이너 작업을 수행하는 방법을 설명합니다.

1.5. RED HAT OPENSTACK PLATFORM에서 CEPH STORAGE 사용

RHOSP(Red Hat OpenStack Platform)를 사용하는 대규모 조직에서는 일반적으로 수많은 클라이언트에 서비스를 제공합니다. 각 OpenStack 클라이언트에는 블록 스토리지 리소스 사용에 대한 자체 요구 사항이 있을 수 있습니다. glance(images), cinder(volumes) 및 nova(Compute)를 단일 노드에 배포할 경우 클라이언트 수가 수천 개인 대규모 배포에서 관리가 불가능해질 수 있습니다. OpenStack을 확장하면 이 문제가 해결됩니다.

하지만 RHOSP 스토리지 계층을 수십 테라바이트에서 페타바이트(또는 엑사바이트) 스토리지로 확장할 수 있도록 Red Hat Ceph Storage와 같은 솔루션을 사용하여 스토리지 계층을 가상화해야 한다는 실질적인 요구도 존재합니다. Red Hat Ceph Storage는 상용 하드웨어에서 실행되는 동안 이 스토리지 가상화 계층에 고가용성 및 고성능을 제공합니다. 가상화로 인해 성능이 저하되는 것처럼 보일 수도 있지만, Ceph 스트라이프는 클러스터에서 오브젝트인 장치 이미지를 차단하므로 큰 Ceph 블록 장치 이미지가 독립 실행형 디스크보다 성능이 뛰어납니다. 또한 Ceph 블록 장치는 성능 향상을 위해 캐싱, copy-on-write 복제 및 copy-on-read 복제를 지원합니다.

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

참고

멀티 아키텍처 클라우드의 경우 Red Hat에서는 사전 설치된 Ceph 구현 또는 외부 Ceph 구현만 지원합니다. 자세한 내용은 Integrating an Overcloud with an Existing Red Hat Ceph Cluster 및 Configuring the CPU architecture for the overcloud 를 참조하십시오.

2장. 언더클라우드 계획

언더클라우드에 director를 구성하고 설치하기 전에 언더클라우드 호스트를 계획하여 특정 요구 사항을 충족해야 합니다.

2.1. 컨테이너화된 언더클라우드

언더클라우드는 오버클라우드라는 최종 RHOSP(Red Hat OpenStack Platform) 환경의 구성, 설치, 관리를 제어하는 노드입니다. 언더클라우드 자체가 OpenStack Platform 구성 요소를 컨테이너 형태로 사용하여 director라는 툴셋을 생성합니다. 즉, 언더클라우드는 레지스트리 소스에서 컨테이너 이미지 세트를 가져와서 컨테이너 구성을 생성하고 각 OpenStack Platform 서비스를 컨테이너로 실행합니다. 결과적으로 언더클라우드는 오버클라우드의 생성 및 관리용 툴셋으로 사용할 수 있는 컨테이너화된 서비스 세트를 제공합니다.

언더클라우드와 오버클라우드 모두 컨테이너를 사용하므로, 둘 다 동일한 아키텍처를 사용하여 컨테이너를 가져오고, 구성하고, 실행합니다. 이 아키텍처는 노드 프로비저닝의 경우 OpenStack Orchestration 서비스(heat)를 기반으로 하고, 서비스 및 컨테이너 구성에는 Ansible을 사용합니다. heat와 Ansible을 어느 정도 익혀 두면 오류 발생 시 문제 해결에 도움이 됩니다.

2.2. 언더클라우드 네트워킹 준비

언더클라우드에서는 다음 두 개의 주요 네트워크에 액세스해야 합니다.

- **프로비저닝 또는 컨트롤 플레인 네트워크**: director에서 노드를 프로비저닝하고, Ansible 구성을 실행할 때 SSH를 통해 해당 노드에 액세스하는 데 사용하는 네트워크입니다. 이 네트워크에서는 언더클라우드 노드에서 오버클라우드 노드로의 SSH 액세스도 가능합니다. 언더클라우드에는 이 네트워크에 있는 다른 노드의 인트로스펙션 및 프로비저닝을 위한 DHCP 서비스가 포함되어 있으므로, 이 네트워크에 다른 DHCP 서비스가 존재해서는 안 됩니다. 이 네트워크의 인터페이스는 director에서 구성합니다.

- **외부 네트워크**: OpenStack Platform 리포지토리, 컨테이너 이미지 소스, DNS 서버 또는 NTP 서버와 같은 기타 서버에 액세스할 수 있습니다. 워크스테이션에서 정상적으로 언더클라우드에 액세스할 때 이 네트워크를 사용합니다. 외부 네트워크에 액세스하려면 언더클라우드의 인터페이스를 수동으로 설정해야 합니다.

언더클라우드에는 최소 두 개의 1Gbps 네트워크 인터페이스 카드가 필요합니다. 하나는 **프로비저닝 또는 컨트롤 플레인 네트워크**용이고, 다른 하나는 **외부 네트워크**용입니다.

네트워크를 계획할 때 다음 지침을 검토하십시오.

- 데이터 플레인에는 하나의 네트워크를 프로비저닝 및 컨트롤 플레인과 다른 네트워크를 사용할 것을 권장합니다. OVS 브리지 상단에 프로비저닝 및 컨트롤 플레인 네트워크를 만들지 마십시오.

- 프로비저닝 및 컨트롤 플레인 네트워크는 Linux 본딩 또는 개별 인터페이스에서 구성할 수 있습니다. Linux 본딩을 사용하는 경우 이를 active-backup 본딩 유형으로 구성합니다.

    - 컨트롤러 이외의 노드에서는 프로비저닝 및 컨트롤 플레인 네트워크에서는 트래픽 양이 상대적으로 낮으며 높은 대역폭 또는 로드 밸런싱이 필요하지 않습니다.

    - 컨트롤러에서 프로비저닝 및 컨트롤 플레인 네트워크에는 추가 대역폭이 필요합니다. 대역폭을 늘리는 이유는 컨트롤러에서 다른 역할에서 많은 노드를 제공하기 때문입니다. 환경에 자주 변경되는 경우에도 더 많은 대역폭이 필요합니다.
      최상의 성능을 위해 컴퓨팅 노드가 50개 이상인 컨트롤러또는 4개 이상의 베어 메탈 노드가 동시에 프로비저닝되는 경우비 컨트롤러 노드의 인터페이스보다 4-10배의 대역폭이 있어야 합니다.

- 50개 이상의 Overcloud 노드를 프로비저닝하는 경우 언더클라우드에 프로비저닝 네트워크에 대한 대역폭 연결이 높아야 합니다.
- 워크스테이션의 director 머신에 액세스하는 데 사용하는 것과 동일한 프로비저닝 또는 컨트롤 플레인 NIC를 사용하지 마십시오. director 설치는 프로비저닝 NIC를 사용하여 브릿지를 생성하고 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.
  - 인트로스펙션 중 프로비저닝 네트워크에 연결된 노드당 임시 IP 주소를 1개 이상 포함합니다.
  - 배포 중 프로비저닝 네트워크에 연결된 노드당 영구 IP 주소를 1개 이상 포함합니다.
  - 프로비저닝 네트워크에서 오버클라우드 고가용성 클러스터의 가상 IP용으로 추가 IP 주소를 포함합니다.
  - 환경 확장을 위해 이 범위 내에 추가 IP 주소를 포함합니다.
- 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.

2.3. 환경 규모 결정

언더클라우드를 설치하기 전에 환경 규모를 결정합니다. 환경을 계획할 때 다음 요소를 고려하십시오.

오버클라우드에 배포할 노드 수

언더클라우드는 오버클라우드 내의 각 노드를 관리합니다. 오버클라우드 노드 프로비저닝에는 언더클라우드의 리소스가 사용됩니다. 모든 오버클라우드 노드를 적절히 프로비저닝하고 제어하려면 언더클라우드에 충분한 리소스를 제공해야 합니다.

언더클라우드 플랫폼에서 동시에 수행할 작업 수

언더클라우드의 OpenStack 서비스에서는 대부분 작업자 세트를 사용합니다. 각 작업자는 해당 서비스와 관련된 작업을 수행합니다. 여러 작업자를 사용하면 동시에 작업을 수행할 수 있습니다. 언더클라우드의 기본 작업자 수는 언더클라우드의 전체 CPU 스레드 수를 반으로 나눈 값입니다. 이 경우 스레드 수는 CPU 코어 수에 하이퍼 스레딩 값을 곱한 값입니다. 예를 들어 언더클라우드의 CPU에 16개의 스레드가 있는 경우 director 서비스는 기본적으로 8개 작업자를 생성합니다. director는 기본적으로 최소 및 최대 한도 세트도 사용합니다.

서비스	최소	최대
OpenStack Orchestration(heat)	4	24
기타 모든 서비스	2	12

언더클라우드에는 다음과 같은 최소 CPU 및 메모리 요구 사항이 있습니다.

- Intel 64 또는 AMD64 CPU 확장 기능을 지원하는 8스레드 64비트 x86 프로세서. 언더클라우드 서비스당 작업자 4개가 제공됩니다.

- 최소 24GB의 RAM
    - **ceph-ansible** 플레이북은 언더클라우드에서 배포된 호스트 10개당 1GB RSS(Resident Set Size)를 사용합니다. 배포에서 새로운 Ceph 클러스터 또는 기존 Ceph 클러스터를 사용하려면 그에 따라 언더클라우드 RAM을 프로비저닝해야 합니다.

다수의 작업자를 사용하려면 다음 권장 사항에 따라 언더클라우드의 vCPU 및 메모리를 늘리십시오.

- **최소값:** 각 스레드에 1.5GB 메모리를 사용합니다. 예를 들어 48개 스레드가 있는 머신은 heat 작업자 24개 및 기타 서비스 작업자 12개에 해당하는 최소 사양으로 72GB의 RAM이 필요합니다.
- **권장 사항:** 각 스레드에 3GB 메모리를 사용합니다. 예를 들어 48개 스레드가 있는 머신은 heat 작업자 24개 및 기타 서비스 작업자 12개에 해당하는 권장 사양으로 144GB의 RAM이 필요합니다.

2.4. 언더클라우드 디스크 크기 조정

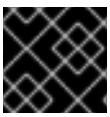
언더클라우드 디스크 최소 권장 크기는 root 디스크에서 **100GB**의 디스크 여유 공간입니다.

- 20GB: 컨테이너 이미지에 사용
- 10GB: 노드 프로비저닝 프로세스 중 QCOW2 이미지 변환 및 캐싱에 사용
- 70GB 이상: 일반 용도, 로깅, 매트릭스, 확장에 사용

2.5. 가상화 지원

Red Hat은 다음 플랫폼에서 가상화된 언더클라우드만 지원합니다.

플랫폼	참고
Kernel-based Virtual Machine (KVM)	인증된 하이퍼바이저에 나열된 Red Hat Enterprise Linux 8.4에서 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

하이퍼바이저가 Red Hat Enterprise Linux 8.4 게스트를 지원하는지 확인합니다.

가상 머신 요구사항

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

네트워크 고려 사항

네트워크 고려 사항

전원 관리

언더클라우드 VM(가상 머신)은 오버클라우드 노드의 전원 관리 장치에 액세스해야 합니다. 이는 노드를 등록할 때 **pm_addr** 매개변수에 설정된 IP 주소입니다.

프로비저닝 네트워크

프로비저닝 네트워크 **ctlplane** 에 사용되는 NIC에는 오버클라우드 베어 메탈 노드의 NIC에 DHCP 요청을 브로드캐스트 및 제공하는 기능이 필요합니다. VM의 NIC를 베어 메탈 NIC와 동일한 네트워크에 연결하는 브릿지를 만듭니다.

알 수 없는 주소의 트래픽 허용

언더클라우드를 차단하는 하이퍼바이저가 알 수 없는 주소에서 트래픽을 전송하지 못하도록 가상 언더클라우드 하이퍼바이저를 구성해야 합니다. 구성은 가상 언더클라우드에 사용하는 플랫폼에 따라 다릅니다.

- Red Hat Enterprise Virtualization: **anti-mac-spoofing** 매개변수를 비활성화합니다.
- VMware ESX 또는 ESXi:
  - IPv4 **ctlplane** 네트워크에서 다음을 수행합니다. 위조된 전송을 허용합니다.
  - IPv6 **ctlplane** 네트워크에서 다음을 수행합니다. 위조된 전송, MAC 주소 변경 및 무차별 모드 작동을 허용합니다.
    VMware ESX 또는 ESXi 구성 방법에 대한 자세한 내용은 VMware 문서 웹 사이트에서 vSphere Standard Switches 보안을 참조하십시오.

해당 설정을 적용한 후 director VM의 전원을 껐다 켜야 합니다. VM을 재부팅하는 것만으로는 부족합니다.

2.6. 문자 인코딩 구성

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.

2.7. 프록시를 사용하여 언더클라우드를 실행할 때 고려 사항

프록시로 언더클라우드를 실행하는 경우 특정 제한 사항이 있으며, Red Hat Satellite를 레지스트리 및 패키지 관리에 사용하는 것이 좋습니다.

그러나 프록시를 사용하는 환경이라면 다음 고려 사항을 검토하여 Red Hat OpenStack Platform의 여러 부분을 프록시와 통합하는 다양한 구성 방법 및 각 방법의 제한 사항을 효과적으로 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** 의 최대 길이는 **pam_env** PAM 모듈의 고정된 크기 버퍼로 인해 1024자입니다.
- 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.

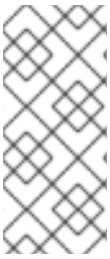
- 이 방법은 Red Hat Subscription Manager에만 프록시 지원을 제공합니다.
- Red Hat Subscription Manager 프록시 구성 값은 시스템 전체 환경 변수에 대해 설정된 모든 값을 재정의합니다.

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.

2.8. 언더클라우드 리포지토리

RHOSP(Red Hat OpenStack Platform) 16.2는 Red Hat Enterprise Linux 8.4에서 실행됩니다. 그러므로 해당 리포지토리의 콘텐츠를 해당하는 Red Hat Enterprise Linux 버전에 고정해야 합니다.



참고

Red Hat Satellite를 사용하여 리포지토리를 동기화하는 경우 특정 버전의 Red Hat Enterprise Linux 리포지토리를 활성화할 수 있습니다. 그러나 리포지토리 레이블은 선택한 버전에도 동일하게 유지됩니다. 예를 들어 8.4 버전의 BaseOS 리포지토리를 활성화하면 리포지토리 이름에 활성화된 특정 버전이 포함되지만 리포지토리 레이블은 여전히 **rhel-8-for-x86_64-baseos-eus-rpms** 입니다.

주의

여기에 지정된 리포지토리만 지원됩니다. 아래 표에 나열되지 않은 제품이나 리포지토리는 별도로 권장되지 않는 한 활성화하지 마십시오. 그러지 않으면 패키지 종속성 문제가 발생할 수 있습니다. EPEL(Extra Packages for Enterprise Linux)을 활성화하지 마십시오.

코어 리포지토리

다음 표에는 언더클라우드 설치에 사용되는 코어 리포지토리가 나와 있습니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - BaseOS(RPM) EUS(Extended Update Support)	**rhel-8-for-x86_64-baseos-eus-rpms**	x86_64 시스템용 기본 운영 체제 리포지토리입니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - AppStream(RPM)	**rhel-8-for-x86_64-appstream-eus-rpms**	Red Hat OpenStack Platform 종속성을 포함합니다.
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다. 컨트롤러 노드 고가용성에 사용됩니다.
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64(RPM)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux입니다. 최신 버전의 Ansible을 제공하는 데 사용됩니다.
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-<satellite_version>-for-rhel-8-x86_64-rpms**	Red Hat Satellite 6으로 호스트를 관리하는 툴입니다. 여기서 **<satellite_version>** 은 사용하는 Red Hat Satellite Server 버전입니다.
Red Hat OpenStack Platform 16.2 for RHEL 8(RPM)	**openstack-16.2-for-rhel-8-x86_64-rpms**	Red Hat OpenStack Platform 코어 리포지토리로 Red Hat OpenStack Platform director용 패키지가 포함됩니다.
Red Hat Fast Datapath for RHEL 8(RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform용 OVS(Open vSwitch) 패키지를 제공합니다.

Ceph 리포지토리

다음 표에는 언더클라우드의 Ceph Storage 관련 리포지토리가 나열되어 있습니다.

이름	리포지토리	요구 사항 설명
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPM)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	노드가 Ceph Storage 클러스터와 통신할 수 있는 툴을 제공합니다. 오버클라우드에서 Ceph Storage를 사용하거나 기존 Ceph Storage 클러스터와 통합하려는 경우 언더클라우드에 이 리포지토리의 **ceph-ansible** 패키지가 필요합니다.

IBM POWER 리포지토리

다음 표에는 POWER PC 아키텍처의 RHOSP용 리포지토리 목록이 나와 있습니다. 코어 리포지토리의 리포지토리 대신 이 리포지토리를 사용하십시오.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux for IBM Power, little endian - BaseOS(RPM)	**rhel-8-for-ppc64le-baseos-rpms**	ppc64le 시스템용 기본 운영 체제 리포지토리입니다.
Red Hat Enterprise Linux 8 for IBM Power, little endian - AppStream(RPM)	**rhel-8-for-ppc64le-appstream-rpms**	Red Hat OpenStack Platform 종속 패키지를 포함합니다.
Red Hat Enterprise Linux 8 for IBM Power, little endian - High Availability(RPM)	**rhel-8-for-ppc64le-highavailability-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다. 컨트롤러 노드 고가용성에 사용됩니다.
Red Hat Fast Datapath for RHEL 8 IBM Power, little endian(RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	OpenStack Platform용 OVS(Open vSwitch) 패키지를 제공합니다.
Red Hat Ansible Engine 2.9 for RHEL 8 IBM Power, little endian(RPM)	**ansible-2.9-for-rhel-8-ppc64le-rpms**	Ansible Engine for Red Hat Enterprise Linux입니다. 최신 버전의 Ansible을 제공합니다.
Red Hat OpenStack Platform 16.2 for RHEL 8(RPM)	**openstack-16.2-for-rhel-8-ppc64le-rpms**	ppc64le 시스템용 Red Hat OpenStack Platform 코어 리포지토리입니다.

3장. DIRECTOR 설치 준비

director를 설치하고 구성하려면 일부 준비 작업을 수행하여 언더클라우드를 Red Hat Customer Portal 또는 Red Hat Satellite 서버에 등록하고, director 패키지를 설치한 후 설치 중에 컨테이너 이미지를 가져오도록 컨테이너 이미지 소스를 설정해야합니다.

3.1. 언더클라우드 준비

director를 설치하려면 먼저 호스트 머신에서 몇 가지 기본 설정을 완료해야 합니다.

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 ~]$
```

6. 시스템 이미지 및 heat 템플릿용 디렉터리를 생성합니다.

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

director는 시스템 이미지와 heat 템플릿을 사용하여 오버클라우드 환경을 생성합니다. 로컬 파일 시스템 구성에 도움이 되도록 이러한 디렉터리를 생성하는 것이 좋습니다.

7. 언더클라우드의 기본 및 전체 호스트 이름을 확인합니다.

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

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

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

8. 언더클라우드 호스트의 정규화된 도메인 이름(FQDN)을 확인할 수 있는 DNS 서버를 사용하지

않는 경우 **/etc/hosts**를 편집하고 시스템 호스트 이름에 대한 항목을 포함합니다. **/etc/hosts**의 IP 주소는 언더클라우드 공용 API에 사용할 주소와 일치해야 합니다. 예를 들어 시스템이 FQDN으로 **undercloud.example.com**을 사용하고 IP 주소로 **10.0.0.1**을 사용하는 경우 **/etc/hosts** 파일에 다음 행을 추가합니다.

```
10.0.0.1  undercloud.example.com undercloud
```

9. 오버클라우드 또는 해당 ID 공급자가 아닌 별도의 도메인에 Red Hat OpenStack Platform director를 배치할 계획인 경우 /etc/resolv.conf에 추가 도메인을 추가해야 합니다.

```
search overcloud.com idp.overcloud.com
```

3.2. 언더클라우드 등록 및 서브스크립션 연결

director를 설치하기 전에 **subscription-manager**를 실행하여 언더클라우드를 등록하고 유효한 Red Hat OpenStack Platform 서브스크립션을 연결해야 합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. Red Hat Content Delivery Network 또는 Red Hat Satellite에 시스템을 등록합니다. 예를 들어 다음 명령을 실행하여 시스템을 콘텐츠 전송 네트워크에 등록합니다. 메시지가 표시되면 고객 포털 사용자 이름과 암호를 입력합니다.

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

3. RHOSP(Red Hat OpenStack Platform) director의 인타이틀먼트 풀 ID를 검색합니다.

```
[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
```

4. **Pool ID** 값을 찾아서 Red Hat OpenStack Platform 16.2 인타이틀먼트를 연결합니다.

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

5. 언더클라우드를 Red Hat Enterprise Linux 8.4에 고정합니다.

```
$ sudo subscription-manager release --set=8.4
```

3.3. 언더클라우드용 리포지토리 활성화

언더클라우드에 필요한 리포지토리를 활성화하고 시스템 패키지를 최신 버전으로 업데이트합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. 기본 리포지토리를 모두 비활성화하고 필수 Red Hat Enterprise Linux 리포지토리를 활성화합니다.

```
[stack@director ~]$ sudo subscription-manager repos --disable=*
[stack@director ~]$ sudo subscription-manager repos \
--enable=rhel-8-for-x86_64-baseos-eus-rpms \
--enable=rhel-8-for-x86_64-appstream-eus-rpms \
--enable=rhel-8-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-8-x86_64-rpms \
--enable=openstack-16.2-for-rhel-8-x86_64-rpms \
--enable=fast-datapath-for-rhel-8-x86_64-rpms \
```

   이러한 리포지토리에는 director 설치에 필요한 패키지가 들어 있습니다.

3. **container-tools** 리포지토리 모듈을 버전 **3.0**으로 설정합니다.

```
[stack@director ~]$ sudo dnf module reset container-tools
[stack@director ~]$ sudo dnf module enable -y container-tools:3.0
```

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

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

3.4. DIRECTOR 패키지 설치

Red Hat OpenStack Platform director와 관련된 패키지를 설치합니다.

절차

1. director 설치 및 설정에 필요한 명령행 툴을 설치합니다.

```
[stack@director ~]$ sudo dnf install -y python3-tripleoclient
```

3.5. CEPH-ANIBLE 설치

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

절차

1. 다음과 같이 Ceph 툴 리포지토리를 활성화합니다.

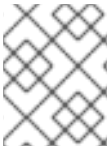
```
[stack@director ~]$ sudo subscription-manager repos --enable=rhceph-4-tools-for-rhel-8-x86_64-rpms
```

2. **ceph-ansible** 패키지를 설치합니다.

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

3.6. 컨테이너 이미지 준비

언더클라우드 설치에는 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 환경 파일이 필요합니다. 컨테이너 이미지를 준비하는 데 사용할 수 있는 이 환경 파일을 생성하고 사용자 지정하십시오.



참고

언더클라우드의 특정 컨테이너 이미지 버전을 구성해야 하는 경우 이미지를 특정 버전에 고정해야 합니다. 자세한 내용은 언더클라우드의 컨테이너 이미지 고정 을 참조하십시오.

절차

1. **stack** 사용자로 언더클라우드 호스트에 로그인합니다.

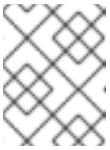
2. 기본 컨테이너 이미지 준비 파일을 생성합니다.

```
$ sudo openstack tripleo container image prepare default \
  --local-push-destination \
  --output-env-file containers-prepare-parameter.yaml
```

이 명령은 다음과 같은 추가 옵션을 사용합니다.

- **--local-push-destination**은 언더클라우드의 레지스트리를 컨테이너 이미지의 위치로 설정합니다. 즉 director가 Red Hat Container Catalog에서 필요한 이미지를 가져와서 언더클라우드의 레지스트리로 푸시합니다. director는 이 레지스트리를 컨테이너 이미지 소스로 사용합니다. Red Hat Container Catalog에서 직접 가져오려면 이 옵션을 생략합니다.

- **--output-env-file**은 환경 파일 이름입니다. 이 파일 내용에는 컨테이너 이미지를 준비하는 데 필요한 매개변수가 포함되어 있습니다. 이 경우 파일 이름은 **containers-prepare-parameter.yaml**입니다.



참고

동일한 **containers-prepare-parameter.yaml** 파일을 사용하여 언더클라우드와 오버클라우드의 컨테이너 이미지 소스를 모두 정의할 수 있습니다.

3. **containers-prepare-parameter.yaml**을 요구 사항에 맞게 수정합니다.

3.7. 컨테이너 이미지 준비 매개변수

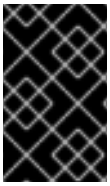
컨테이너를 준비하는 데 필요한 기본 파일(**containers-prepare-parameter.yaml**)에는 **ContainerImagePrepare** heat 매개변수가 포함되어 있습니다. 이 매개변수는 이미지 세트를 준비하기 위한 다양한 설정을 정의합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

각각의 설정에서 하위 매개변수 세트를 통해 사용할 이미지와 해당 이미지의 사용 방법을 정의할 수 있습니다. 다음 표에는 각 **ContainerImagePrepare** 전략에 사용할 수 있는 하위 매개변수에 대한 정보가 나와 있습니다.

매개변수	설명
excludes	전략에서 이미지 이름을 제외하는 정규식 목록입니다.
includes	전략에 포함할 정규식 목록입니다. 기존 이미지와 일치하는 이미지 이름이 하나 이상 있어야 합니다. **includes**를 지정하면 **excludes**는 모두 무시됩니다.
modify_append_tag	대상 이미지 태그에 추가할 문자열입니다. 예를 들어 16.2.3-5.161 태그가 있는 이미지를 가져와서 **modify_append_tag**를 **-hotfix**로 설정하면 director에서 최종 이미지를 16.2.3-5.161-hotfix로 태그합니다.
modify_only_with_labels	수정할 이미지를 필터링하는 이미지 레이블로 이루어진 사전입니다. 이미지가 정의된 레이블과 일치하면 director에서 그 이미지를 수정 프로세스에 포함합니다.
modify_role	이미지를 대상 레지스트리로 푸시하기 전 업로드 중에 실행할 ansible 역할 이름의 문자열입니다.
modify_vars	**modify_role**로 전달할 변수로 이루어진 사전입니다.

매개변수	설명
push_destination	업로드 프로세스 중 이미지를 푸시할 레지스트리의 네임스페이스를 정의합니다. ● **true**로 설정하면 **push_destination**이 해당 호스트 이름을 사용하는 언더클라우드 레지스트리 네임스페이스로 설정되며, 이것이 권장되는 방법입니다. ● **false**로 설정하면 로컬 레지스트리로 푸시되지 않고 노드가 소스에서 직접 이미지를 가져옵니다. ● 사용자 지정 값으로 설정하면 director가 이미지를 외부 로컬 레지스트리로 푸시합니다. Red Hat Container Catalog에서 직접 이미지를 가져오는 동안 프로덕션 환경에서 이 매개변수를 **false**로 설정하면 모든 오버클라우드 노드에서 외부 연결을 통해 Red Hat Container Catalog의 이미지를 동시에 가져옵니다. 이로 인해 대역폭 문제가 발생할 수 있습니다. 컨테이너 이미지를 호스팅하는 Red Hat Satellite 서버에서 이미지를 직접 가져올 때만 **false**를 사용하십시오. **push_destination** 매개변수가 정의되지 않았거나 **false**로 설정되었는데 원격 레지스트리에 인증이 필요한 경우, **ContainerImageRegistryLogin** 매개변수를 **true**로 설정하고 **ContainerImageRegistryCredentials** 매개변수가 있는 인증 정보를 포함하십시오.
pull_source	원본 컨테이너 이미지를 가져온 소스 레지스트리입니다.
set	초기 이미지를 가져올 위치를 정의하는 **key: value** 정의로 이루어진 사전입니다.
tag_from_label	지정된 컨테이너 이미지 메타데이터 레이블의 값을 사용하여 모든 이미지에 대한 태그를 생성하고 해당 태그가 지정된 이미지를 가져옵니다. 예를 들어 **tag_from_label: {version}-{release}**를 설정하면 director는 **version** 및 **release** 레이블을 사용하여 새 태그를 구성합니다. 한 컨테이너에서 **version**을 16.2.3으로 설정하고 **release**를 **5.161**로 설정할 수 있으므로 태그는 16.2.3-5.161이 됩니다. director는 **set** 사전에 **tag**가 정의되지 않은 경우에만 이 매개변수를 사용합니다.



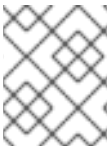
중요

이미지를 언더클라우드로 내보내는 경우 **push _destination 대신 push_destination: true 를 사용합니다. UNDERCLOUD_IP:PORT**. **push_destination: true** 방식은 IPv4 및 IPv6 주소 모두에서 일관성 수준을 제공합니다.

set 매개변수에 여러 **key: value** 정의를 사용할 수 있습니다.

키	설명
ceph_image	Ceph Storage 컨테이너 이미지의 이름입니다.
ceph_namespace	Ceph Storage 컨테이너 이미지의 네임스페이스입니다.
ceph_tag	Ceph Storage 컨테이너 이미지의 태그입니다.
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	Ceph Storage Alert Manager 컨테이너 이미지의 이름, 네임스페이스 및 태그입니다.
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	Ceph Storage Grafana 컨테이너 이미지의 이름, 네임스페이스 및 태그입니다.
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	Ceph Storage Node Exporter 컨테이너 이미지의 이름, 네임스페이스 및 태그입니다.
ceph_prometheus_image **ceph_prometheus_namespace** **ceph_prometheus_tag**	Ceph Storage Prometheus 컨테이너 이미지의 이름, 네임스페이스 및 태그입니다.
name_prefix	각 OpenStack 서비스 이미지의 접두사입니다.
name_suffix	각 OpenStack 서비스 이미지의 접미사입니다.
namespace	각 OpenStack 서비스 이미지의 네임스페이스입니다.
neutron_driver	사용할 OpenStack Networking (Neutron) 컨테이너를 결정하는 데 사용할 드라이버입니다. null 값을 사용하여 표준 **neutron-server** 컨테이너를 설정합니다. OVN 기반 컨테이너를 사용하려면 **ovn**으로 설정합니다.

키	설명
tag	소스의 모든 이미지에 대한 특정 태그를 설정합니다. 정의되지 않은 경우 director는 Red Hat OpenStack Platform 버전 번호를 기본값으로 사용합니다. 이 매개변수가 **tag_from_label** 값보다 우선합니다.



참고

컨테이너 이미지는 Red Hat OpenStack Platform 버전을 기반으로 멀티 스트림 태그를 사용합니다. 즉, 더 이상 **latest** 태그가 없음을 의미합니다.

3.8. 컨테이너 이미지 태그 지침

Red Hat Container Registry는 특정 버전 형식을 사용하여 모든 Red Hat OpenStack Platform 컨테이너 이미지에 태그를 지정합니다. 이 형식은 **version-release**인 각 컨테이너의 레이블 메타데이터를 따릅니다.

버전

Red Hat OpenStack Platform의 주요 및 마이너 버전에 해당합니다. 이러한 버전은 하나 이상의 릴리스를 포함하는 스트림으로 작동합니다.

릴리스

버전 스트림 내 특정 컨테이너 이미지 버전의 릴리스에 해당합니다.

예를 들어 최신 버전의 Red Hat OpenStack Platform이 16.2.3이고 컨테이너 이미지의 릴리스가 **5.161**인 경우 컨테이너 이미지의 결과 태그는 16.2.3-5.161입니다.

Red Hat Container Registry에서는 해당 컨테이너 이미지 버전의 최신 릴리스에 연결되는 메이저 및 마이너 **version** 버전 태그 세트도 사용합니다. 예를 들어 16.2 및 16.2.3은 모두 16.2.3 컨테이너 스트림의 최신 **release**에 연결됩니다. 16.2의 새 마이너 릴리스가 발생하면 16.2.3 태그가 16.2.3 스트림 내의 최신 **release**에 계속 연결되는 반면 16.2 태그는 최신 **release**에 연결됩니다.

ContainerImagePrepare 매개변수에는 다운로드할 컨테이너 이미지를 결정하는 데 사용할 두 개의 하위 매개변수가 포함되어 있습니다. 이러한 하위 매개변수는 **set** 사전 내의 **tag** 매개변수와 **tag_from_label** 매개변수입니다. 다음 지침을 사용하여 **tag** 또는 **tag_from_label**을 사용할지 여부를 결정합니다.

- **tag**의 기본값은 OpenStack Platform 버전의 주요 버전입니다. 이 버전의 경우 16.2입니다. 이는 항상 최신 마이너 버전 및 릴리스에 해당합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 16.2
      ...
```

- OpenStack Platform 컨테이너 이미지의 특정 마이너 버전으로 변경하려면 태그를 마이너 버전으로 설정합니다. 예를 들어 16.2.2로 변경하려면 **tag**를 16.2.2로 설정합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
```

```
    ...
    tag: 16.2.2
    ...
```

- **tag**를 설정하면 director는 설치 및 업데이트 중에 **tag**에 설정된 버전의 최신 컨테이너 이미지 **release**를 항상 다운로드합니다.
- **tag**를 설정하지 않으면 director는 최신 주요 버전과 함께 **tag_from_label** 값을 사용합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      # tag: 16.2
      ...
    tag_from_label: '{version}-{release}'
```

- **tag_from_label** 매개변수는 Red Hat Container Registry에서 검사하는 최신 컨테이너 이미지 릴리스의 레이블 메타데이터에서 태그를 생성합니다. 예를 들어 특정 컨테이너의 레이블에서 다음 **version** 및 **release** 메타데이터를 사용할 수 있습니다.

```
"Labels": {
  "release": "5.161",
  "version": "16.2.3",
  ...
}
```

- **tag_from_label**의 기본값은 **{version}-{release}**로, 각 컨테이너 이미지의 버전 및 릴리스 메타데이터 레이블에 해당합니다. 예를 들어 컨테이너 이미지에 **version**용으로 16.2.3이 설정되어 있고 **release**용으로 5.161이 설정된 경우 컨테이너 이미지의 결과 태그는 16.2.3-5.161입니다.
- **tag** 매개변수는 항상 **tag_from_label** 매개변수보다 우선합니다. **tag_from_label**을 사용하려면 컨테이너 준비 구성에서 **tag** 매개변수를 생략합니다.
- **tag**와 **tag_from_label**의 주요 차이점은 director가 **tag**를 사용하여 주요 또는 마이너 버전 태그를 기반으로만 이미지를 가져온다는 것입니다. 이 태그는 버전 스트림 내의 최신 이미지 릴리스에 대한 Red Hat Container Registry 링크인 반면 director는 **tag_from_label**을 사용하여 director가 태그를 생성하고 해당 이미지를 가져올 수 있도록 각 컨테이너 이미지의 메타데이터 검사를 수행합니다.

3.9. 개인 레지스트리에서 컨테이너 이미지 가져오기

registry.redhat.io 레지스트리에는 이미지에 액세스하여 가져오기 위한 인증이 필요합니다. **registry.redhat.io** 및 기타 개인 레지스트리로 인증하려면 **containers-prepare-parameter.yaml** 파일에 **ContainerImageRegistryCredentials** 및 **ContainerImageRegistryLogin** 매개변수를 포함합니다.

ContainerImageRegistryCredentials

일부 컨테이너 이미지 레지스트리는 이미지에 액세스하기 위해 인증이 필요합니다. 이 경우 **container-prepare-parameter.yaml** 환경 파일에서 **ContainerImageRegistryCredentials** 매개변수를 사용합니다. **ContainerImageRegistryCredentials** 매개변수는 개인 레지스트리 URL에 따라 여러 키를 사용합니다. 각 개인 레지스트리 URL은 고유한 키와 값 쌍을 사용하여 사용자 이름(키)과 암호(값)를 정의합니다. 이런 방법으로 여러 개인 레지스트리의 인증 정보를 지정할 수 있습니다.

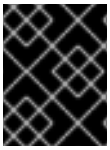
```
parameter_defaults:
```

```
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      my_username: my_password
```

예제에서 **my_username** 및 **my_password**를 사용자 인증 정보로 교체합니다. 개별 사용자 인증 정보를 사용하는 대신, 레지스트리 서비스 계정을 생성하고 해당 인증 정보를 사용하여 **registry.redhat.io** 콘텐츠에 액세스하는 것이 좋습니다.

여러 레지스트리에 대한 인증 세부 정보를 지정하려면 **ContainerImageRegistryCredentials**의 각 레지스트리에 대해 여러 개의 키 쌍 값을 설정합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  - push_destination: true
    set:
      namespace: registry.internalsite.com/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
    registry.internalsite.com:
      myuser2: '0th3rp@55w0rd!'
    '192.0.2.1:8787':
      myuser3: '@n0th3rp@55w0rd!'
```



중요

기본 **ContainerImagePrepare** 매개변수는 인증이 필요한 **registry.redhat.io**에서 컨테이너 이미지를 가져옵니다.

자세한 내용은 Red Hat Container Registry Authentication 을 참조하십시오.

ContainerImageRegistryLogin

ContainerImageRegistryLogin 매개변수는 오버클라우드 노드 시스템이 컨테이너 이미지를 가져오기 위해 원격 레지스트리에 로그인해야 하는지 여부를 제어하는 데 사용합니다. 이러한 상황은 오버클라우드 노드에서 이미지를 호스팅하는 데 언더클라우드를 사용하는 대신 이미지를 직접 가져오도록 할 때 발생합니다.

지정된 설정에 대해 **push_destination**이 구성되지 않은 경우 **ContainerImageRegistryLogin**을 **true**로 설정해야 합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
```

```
    set:
      namespace: registry.redhat.io/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: true
```

하지만 오버클라우드 노드에서 **ContainerImageRegistryCredentials**에 정의된 레지스트리 호스트에 네트워크로 연결할 수 없고 **ContainerImageRegistryLogin**을 **true**로 설정하는 경우, 로그인할 때 배포에 실패할 수 있습니다. 오버클라우드 노드에서 **ContainerImageRegistryCredentials**에 정의된 레지스트리 호스트에 네트워크로 연결할 수 없고 **push_destination**을 **true**로 설정하고 **ContainerImageRegistryLogin**을 **false**로 설정하여 오버클라우드 노드가 언더클라우드에서 이미지를 가져오도록 합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: false
```

3.10. 이미지 준비 항목 계층화

ContainerImagePrepare 매개변수의 값은 YAML 목록입니다. 즉, 여러 항목을 지정할 수 있습니다.

다음 예제에서는 두 개의 항목을 지정하는 경우를 설명합니다. 이때 director는 **16.2.1-hotfix** 로 태그된 버전을 사용하는 **nova-api** 이미지를 제외하고 모든 이미지의 최신 버전을 사용합니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - tag_from_label: "{version}-{release}"
    push_destination: true
    excludes:
    - nova-api
    set:
      namespace: registry.redhat.io/rhosp-rhel8
      name_prefix: openstack-
      name_suffix: ''
      tag:16.2
  - push_destination: true
    includes:
    - nova-api
    set:
      namespace: registry.redhat.io/rhosp-rhel8
      tag: 16.2.1-hotfix
```

includes 및 **excludes** 매개 변수는 정규식을 사용하여 각 항목에 대한 이미지 필터링을 제어합니다. **includes** 설정과 일치하는 이미지가 **excludes**와 일치하는 이미지보다 우선합니다. 일치하는 이미지로 간주되려면 이미지 이름이 **includes** 또는 **excludes** 정규식 값과 일치해야 합니다.

Block Storage(cinder) 드라이버에 플러그인이라는 공급된 cinder-volume 이미지가 필요한 경우에도 유사한 기술이 사용됩니다. 블록 스토리지 드라이버에 플러그인이 필요한 경우 *Advanced Overcloud Customization* 가이드의 벤더 플러그인 배포를 참조하십시오.

3.11. CEPH STORAGE 컨테이너 이미지 제외

기본 오버클라우드 역할 구성에서는 기본 Controller, Compute 및 Ceph Storage 역할을 사용합니다. 그러나 기본 역할 구성을 사용하여 Ceph Storage 노드 없이 오버클라우드를 배포하는 경우 이미지가 기본 구성의 일부로 포함되어 있으므로 director는 Red Hat Container Registry에서 Ceph Storage 컨테이너 이미지를 계속 가져옵니다.

오버클라우드에 Ceph Storage 컨테이너가 필요하지 않은 경우 Red Hat Container Registry에서 Ceph Storage 컨테이너 이미지를 가져오지 않도록 director를 구성할 수 있습니다.

절차

1. **containers-prepare-parameter.yaml** 파일을 편집하여 Ceph Storage 컨테이너를 제외합니다.

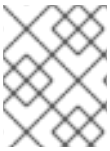
```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    excludes:
      - ceph
      - prometheus
    set:
      …
```

   excludes 매개변수는 정규식을 사용하여 **ceph** 또는 **prometheus** 문자열이 포함된 컨테이너 이미지를 제외합니다.

2. **containers-prepare-parameter.yaml** 파일을 저장합니다.

3.12. 준비 과정에서 이미지 수정

이미지 준비 과정에서 이미지를 수정한 다음, 수정된 이미지로 오버클라우드를 즉시 배포할 수 있습니다.



참고

RHOSP(Red Hat OpenStack Platform) director는 Ceph 컨테이너가 아닌 RHOSP 컨테이너를 준비하는 동안 이미지 수정을 지원합니다.

이미지 수정 시나리오는 다음과 같습니다.

- 지속적 통합 파이프라인에서 배포 전 테스트 중인 변경 사항으로 이미지가 수정되는 경우입니다.
- 개발 워크플로우 중에 테스트 및 개발을 위해 로컬 변경 사항을 배포해야 하는 경우입니다.
- 변경 사항을 배포해야 하지만 이미지 빌드 파이프라인을 통해 사용할 수 없는 경우입니다. 예를 들어 독점 애드온 또는 긴급 수정 사항을 추가할 수 있습니다.

준비 과정에서 이미지를 수정하려면, 수정할 각 이미지에 대해 Ansible 역할을 호출합니다. 이 역할은 소스 이미지를 가져와서 요청된 변경을 수행한 다음 그 결과를 태그합니다. prepare 명령으로 이미지를 대상 레지스트리로 푸시하고 수정된 이미지를 참조하도록 heat 매개변수를 설정할 수 있습니다.

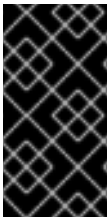
Ansible 역할 **tripleo-modify-image**는 요청된 역할 인터페이스를 준수하고 수정 사용 사례에 필요한 작업을 수행합니다. **ContainerImagePrepare** 매개변수의 수정 관련 키를 사용하여 수정을 제어합니다.

- **modify_role**은 수정할 각 이미지에 대해 호출할 Ansible 역할을 지정합니다.
- **modify_append_tag**는 소스 이미지 태그의 끝에 문자열을 추가합니다. 이렇게 하면 결과 이미지가 수정되었음을 알 수 있습니다. **push_destination** 레지스트리에 수정된 이미지가 이미 포함되어 있을 경우 이 매개변수를 사용하여 수정을 생략할 수 있습니다. 이미지를 수정할 때마다 **modify_append_tag**를 변경하십시오.
- **modify_vars**는 역할에 전달할 Ansible 변수로 이루어진 사전입니다.

tripleo-modify-image 역할에서 처리할 사용 케이스를 선택하려면 **tasks_from** 변수를 해당 역할에 필요한 파일로 설정합니다.

이미지를 수정하려면 **ContainerImagePrepare** 항목을 개발하고 테스트하는 동안 추가 옵션 없이 image prepare 명령을 실행하여 이미지가 예상대로 수정되는지 확인합니다.

```
sudo openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```



중요

openstack tripleo container image prepare 명령을 사용하려면 언더클라우드에 실행 중인 **image-serve** 레지스트리가 포함되어야 합니다. 따라서 **image-serve** 레지스트리가 설치되지 않으므로 새 언더클라우드를 설치하기 전에 이 명령을 실행할 수 없습니다. 언더클라우드를 성공적으로 설치한 후 이 명령을 실행할 수 있습니다.

3.13. 컨테이너 이미지의 기존 패키지 업데이트

참고

RHOSP(Red Hat OpenStack Platform) director는 Ceph 컨테이너가 아닌 RHOSP 컨테이너의 컨테이너 이미지의 기존 패키지 업데이트를 지원합니다.

절차

- 다음 예제 **ContainerImagePrepare** 항목은 언더클라우드 호스트의 dnf 리포지토리 구성을 사용하여 컨테이너 이미지의 모든 패키지에서 업데이트됩니다.

```
ContainerImagePrepare:
- push_destination: true
  ...
  modify_role: tripleo-modify-image
  modify_append_tag: "-updated"
  modify_vars:
    tasks_from: yum_update.yml
    compare_host_packages: true
    yum_repos_dir_path: /etc/yum.repos.d
  ...
```

3.14. 컨테이너 이미지에 추가 RPM 파일 설치

컨테이너 이미지에 RPM 파일 디렉터리를 설치할 수 있습니다. 이 기능은 핫픽스, 로컬 패키지 빌드 또는 패키지 리포지토리를 통해 사용할 수 없는 패키지를 설치하는 데 유용합니다.

참고

RHOSP(Red Hat OpenStack Platform) director는 Ceph 컨테이너가 아닌 RHOSP 컨테이너의 컨테이너 이미지에 추가 RPM 파일을 설치할 수 있도록 지원합니다.

절차

- 다음 예제 **ContainerImagePrepare** 항목은 **nova-compute** 이미지에만 일부 핫픽스 패키지를 설치합니다.

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: rpm_install.yml
    rpms_path: /home/stack/nova-hotfix-pkgs
  ...
```

3.15. 사용자 지정 DOCKERFILE을 사용하여 컨테이너 이미지 수정

Dockerfile이 포함된 디렉터리를 지정하여 필요한 변경을 수행할 수 있습니다. **tripleo-modify-image** 역할을 호출하면 **FROM** 지시문을 변경하고 **LABEL** 지시문을 추가하는 **Dockerfile.modified** 파일이 생성됩니다.

참고

RHOSP(Red Hat OpenStack Platform) director는 Ceph 컨테이너가 아닌 RHOSP 컨테이너의 사용자 지정 Dockerfile을 사용하여 컨테이너 이미지 수정을 지원합니다.

절차

1. 다음 예제에서는 **nova-compute** 이미지에 대해 사용자 지정 Dockerfile을 실행합니다.

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: modify_image.yml
    modify_dir_path: /home/stack/nova-custom
  ...
```

2. **/home/stack/nova-custom/Dockerfile** 파일의 예는 다음과 같습니다. **USER** root 지시문을 실행한 후에는 원본 이미지의 기본 사용자로 다시 전환해야 합니다.

```
FROM registry.redhat.io/rhosp-rhel8/openstack-nova-compute:latest

USER "root"

COPY customize.sh /tmp/
RUN /tmp/customize.sh

USER "nova"
```

3.16. 컨테이너 이미지용 SATELLITE 서버 준비

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 관한 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.

참고

다음 절차에서는 **registry.redhat.io**에서 컨테이너 이미지에 액세스하기 위해 인증 정보가 필요합니다. 개별 사용자 인증 정보를 사용하는 대신, 레지스트리 서비스 계정을 생성하고 해당 인증 정보를 사용하여 **registry.redhat.io** 콘텐츠에 액세스하는 것이 좋습니다. 자세한 내용은 "Red Hat Container Registry Authentication" 을 참조하십시오.

절차

1. 모든 컨테이너 이미지 목록을 생성합니다.

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{ .Name }}" | sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-dashboard-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/openshift" | grep ose-prometheus
```

- Ceph를 설치하고 Ceph 대시보드를 활성화하려면 다음과 같은 ose-prometheus 컨테이너가 필요합니다.

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.6
registry.redhat.io/openshift4/ose-prometheus:v4.6
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.6
```

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

3. 다음 **hammer** 명령을 실행하여 Satellite 조직에 새 제품(**OSP Containers**)을 생성합니다.

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

이 사용자 지정 제품에 이미지를 저장합니다.

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

```
$ while read IMAGE; do \
  IMAGE_NAME=$(echo $IMAGE | cut -d"/" -f3 | sed "s/openstack-//g") ; \
  IMAGE_NOURL=$(echo $IMAGE | sed "s/registry.redhat.io\///g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE_NOURL \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGE_NAME ; done < satellite_images
```

5. Ceph Storage 4 컨테이너 이미지를 추가합니다.

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-rhel8
```

참고

Ceph 대시보드를 설치하려면 **hammer repository create** 명령에 **--name rhceph-4-dashboard-rhel8** 을 포함합니다.

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-dashboard-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-dashboard-rhel8
```

6. 컨테이너 이미지를 동기화합니다.

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP 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. 자세한 내용은 *Hammer CLI Guide*의 "Authentication" 섹션을 참조하십시오.

7. Satellite 6 서버에서 콘텐츠 뷰를 사용하는 경우, 새로운 콘텐츠 뷰 버전을 생성하여 이미지를 통합하고 애플리케이션 라이프사이클의 환경에 따라 승격합니다. 이 과정은 대체로 애플리케이션 라이프사이클을 구조화한 방법에 따라 달라집니다. 예를 들어 라이프사이클에 **production**이라는 환경이 있고 해당 환경에서 컨테이너 이미지를 사용할 수 있도록 하려면 컨테이너 이미지가 포함된 콘텐츠 뷰를 생성하고 해당 콘텐츠 뷰를 **production** 환경으로 승격합니다. 자세한 내용은 Managing Content Views를 참조하십시오.

8. **base** 이미지에 사용 가능한 태그를 확인합니다.

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --lifecycle-environment "production" \
  --product "OSP Containers"
```

이 명령을 수행하면 OpenStack Platform 컨테이너 이미지의 태그가 특정 환경의 콘텐츠 뷰에 표시됩니다.

9. 언더클라우드로 돌아가서 Satellite 서버를 소스로 사용하여 이미지를 준비하는 기본 환경 파일을 생성합니다. 다음 예제 명령을 실행하여 환경 파일을 생성합니다.

```
$ sudo openstack tripleo container image prepare default \
  --output-env-file containers-prepare-parameter.yaml
```

- **--output-env-file**은 환경 파일 이름입니다. 이 파일의 콘텐츠에는 언더클라우드의 컨테이너 이미지를 준비하는 데 필요한 매개변수가 포함되어 있습니다. 이 경우 파일 이름은 **containers-prepare-parameter.yaml**입니다.

10. **containers-prepare-parameter.yaml** 파일을 편집하여 다음 매개변수를 수정합니다.

- **push_destination** - 선택한 컨테이너 이미지 관리 전략에 따라 이 값을 **true** 또는 **false**로 설정합니다. 이 매개변수를 **false**로 설정하면 오버클라우드 노드는 Satellite에서 직접 이미지를 가져옵니다. 이 매개변수를 **true**로 설정하면 director가 Satellite에서 언더클라우드 레지스트리로 이미지를 가져오고 오버클라우드는 언더클라우드 레지스트리에서 이미지를 가져옵니다.

- **namespace** - Satellite 서버 레지스트리의 URL 및 포트입니다. Red Hat Satellite의 기본 레지스트리 포트는 443입니다.

- **name_prefix** - 접두사는 Satellite 6 규칙을 기반으로 하며, 콘텐츠 뷰 사용 여부에 따라 달라집니다.

  - 콘텐츠 뷰를 사용하는 경우 구조는 **[org]-[environment]-[content view]-[product]-**입니다. 예를 들면 **acme-production-myosp16-osp_containers-** 입니다.

  - 콘텐츠 뷰를 사용하지 않는 경우 구조는 **[org]-[product]-**입니다. 예를 들면 **acme-osp_containers-** 입니다.

- **ceph_namespace**, **ceph_image**, **ceph_tag** - Ceph Storage를 사용하는 경우 추가 매개변수를 포함하여 Ceph Storage 컨테이너 이미지 위치를 정의합니다. 이제 **ceph_image**에는 Satellite별 접두사가 포함됩니다. 이 접두사는 **name_prefix** 옵션과 동일한 값입니다.

다음 예제 환경 파일에는 Satellite별 매개변수가 포함되어 있습니다.

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp16_1-osp_containers-rhceph-4
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp16_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:443
      neutron_driver: null
      tag: '16.2'
      ...
```

참고

Red Hat Satellite Server에 저장된 특정 컨테이너 이미지 버전을 사용하려면 **태그** 키-값 쌍을 **세트 사전의 특정 버전으로 설정합니다**. 예를 들어 16.2.2 이미지 스트림을 사용하려면 **tag를 설정합니다. 16.2.2 세트** 사전의.

undercloud.conf 구성 파일에 **containers-prepare-parameter.yaml** 환경 파일을 정의해야 합니다. 그렇지 않으면 언더클라우드에서 기본값을 사용합니다.

```
container_images_file = /home/stack/containers-prepare-parameter.yaml
```

4장. 언더클라우드에 DIRECTOR 설치

director를 구성하고 설치하려면 **undercloud.conf** 파일에 적절한 매개변수를 설정하고 undercloud installation 명령을 실행합니다. director를 설치한 후 노드 프로비저닝 중에 베어 메탈 노드에 쓰는 데 사용할 오버클라우드 이미지를 가져옵니다.

4.1. DIRECTOR 구성

director 설치 프로세스에는 director가 **stack** 사용자의 홈 디렉터리에서 읽어오는 **undercloud.conf** 구성 파일의 특정 설정이 필요합니다. 다음 단계에 따라 기본 템플릿을 복사하여 설정 기반으로 사용하십시오.

절차

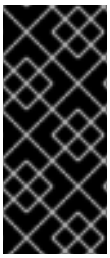
1. 기본 템플릿을 **stack** 사용자의 홈 디렉터리에 복사합니다.

```
[stack@director ~]$ cp \
  /usr/share/python-tripleoclient/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.

4.2. DIRECTOR 설정 매개변수

다음 목록에는 **undercloud.conf** 파일을 설정하기 위한 매개변수 정보가 나와 있습니다. 오류를 방지하려면 모든 매개변수를 관련 섹션에 보관합니다.



중요

최소한 **container_images_file** 매개변수를 컨테이너 이미지 구성이 포함된 환경 파일로 설정해야 합니다. 이 매개변수를 적절한 파일로 올바르게 설정하지 않으면 director가 **ContainerImagePrepare** 매개변수에서 컨테이너 이미지 규칙 세트를 가져올 수 없거나 **ImageRegistryCredentials** 매개변수에서 컨테이너 레지스트리 인증 세부 정보를 가져올 수 없습니다.

기본값

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

additional_architectures

오버클라우드에서 지원하는 추가(커널) 아키텍처 목록입니다. 현재 오버클라우드는 기본 **x86_64** 아키텍처 외에 **ppc64le** 아키텍처를 지원합니다.

참고

ppc64le 지원을 활성화하는 경우 **ipxe_enabled** 를 **False** 로 설정해야 합니다. 여러 CPU 아키텍처를 사용하여 언더클라우드를 구성하는 방법에 대한 자세한 내용은 여러 CPU 아키텍처 오버클라우드 구성을 참조하십시오.

certificate_generation_ca

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

clean_nodes

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

cleanup

임시 파일을 정리합니다. 배포 명령을 실행한 후 배포 중 사용한 임시 파일을 그대로 두려면 **False**로 설정하십시오. 이 매개변수는 생성된 파일을 디버깅하거나 오류가 발생한 경우에 유용합니다.

container_cli

컨테이너 관리를 위한 CLI 툴입니다. 이 매개변수를 **podman**으로 설정해 둡니다. Red Hat Enterprise Linux 8.4는 **podman**만 지원합니다.

container_healthcheck_disabled

컨테이너화된 서비스 상태 점검을 비활성화합니다. 상태 점검을 활성화하고 이 옵션을 **false**로 설정해 두는 것이 좋습니다.

container_images_file

컨테이너 이미지 정보가 포함된 heat 환경 파일입니다. 이 파일은 다음 항목을 포함할 수 있습니다.

- 필요한 모든 컨테이너 이미지에 대한 매개변수
- 필요한 이미지 준비를 수행하는 **ContainerImagePrepare** 매개변수. 일반적으로 이 매개변수가 포함된 파일의 이름은 **containers-prepare-parameter.yaml**입니다.

container_insecure_registries

podman이 사용할 수 있는 비보안 레지스트리 목록입니다. 개인 컨테이너 레지스트리와 같은 다른 소스에서 이미지를 가져오려는 경우 이 매개변수를 사용합니다. 대부분의 경우 언더클라우드가 Satellite에 등록되어 있으면, Red Hat Container Catalog 또는 Satellite 서버에서 컨테이너 이미지를 가져올 수 있는 인증서가 **podman**에 있습니다.

container_registry_mirror

podman에서 사용하도록 구성된 **registry-mirror**(선택 사항)입니다.

custom_env_files

언더클라우드 설치에 추가할 추가 환경 파일입니다.

deployment_user

언더클라우드를 설치하는 사용자입니다. 현재 기본 사용자인 **stack**을 사용하려면 이 매개변수를 설정되지 않은 상태로 두십시오.

discovery_default_driver

자동으로 등록된 노드의 기본 드라이버를 설정합니다. **enable_node_discovery** 매개변수를 활성화해야 하며, 드라이버를 **enabled_hardware_types** 목록에 포함해야 합니다.

enable_ironic; enable_ironic_inspector; enable_mistral; enable_nova; enable_tempest; enable_validations; enable_zaqar

director를 위해 활성화할 코어 서비스를 정의합니다. 이 매개변수를 **true**로 설정된 상태로 두십시오.

enable_node_discovery

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

enable_novajoin

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

enable_routed_networks

라우팅된 컨트롤 플레인 네트워크에 대한 지원을 활성화할지 여부를 정의합니다.

enable_swift_encryption

유휴 시 Swift 암호화를 활성화할지 여부를 정의합니다.

enable_telemetry

언더클라우드에 OpenStack Telemetry 서비스(gnocchi, aodh, panko)를 설치할지 여부를 정의합니다. Telemetry 서비스를 자동으로 설치하고 구성하려면 **enable_telemetry** 매개변수를 **true**로 설정합니다. 기본값은 **false**로, 언더클라우드에서 Telemetry를 비활성화합니다. 이 매개변수는 메트릭 데이터를 사용하는 Red Hat CloudForms 같은 제품을 사용할 때 필요합니다.

주의

RBAC는 모든 구성 요소에서 지원되지 않습니다. 알람 서비스(aodh) 및 Gnocchi는 보안 RBAC 규칙을 고려하지 않습니다.

enabled_hardware_types

언더클라우드를 위해 활성화할 하드웨어 유형 목록입니다.

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.

heat_container_image

사용할 Heat 컨테이너 이미지의 URL입니다. 설정되지 않은 상태로 두십시오.

heat_native

heat-all을 사용하여 호스트 기반 언더클라우드 설정을 실행합니다. **true**로 두십시오.

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 관한 자세한 내용은 언더클라우드에서 hieradata 구성을 참조하십시오.

inspection_extras

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

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_runbench

노드 인트로스펙션 중 벤치마크 집합을 실행합니다. 벤치마크를 활성화하려면 이 매개변수를 **true**로 설정합니다. 등록된 노드의 하드웨어를 검사할 때 벤치마크 분석을 수행하려는 경우 이 옵션이 필요합니다.

ipa_otp

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

ipv6_address_mode

언더클라우드 프로비저닝 네트워크의 IPv6 주소 구성 모드입니다. 다음 목록에 이 매개변수에 사용할 수 있는 값이 나와 있습니다.

- dhcpv6-stateless - RA(라우터 알림)를 사용한 주소 구성 및 DHCPv6을 사용한 선택적 정보입니다.
- dhcpv6-stateful - DHCPv6을 사용한 주소 설정 및 선택적 정보입니다.

ipxe_enabled

iPXE 또는 표준 PXE 사용 여부를 정의합니다. 기본값은 **true**이며, iPXE를 활성화합니다. 표준 PXE를 사용하려면 이 매개변수를 **false**로 설정하십시오. PowerPC 배포 또는 하이브리드 PowerPC 및 x86 배포의 경우 이 값을 **false** 로 설정합니다.

local_interface

director 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: em0: <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 em0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: em1: <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 **em0**을 사용하고, 프로비저닝 NIC는 현재 구성되지 않은 **em1**을 사용합니다. 이 경우에는 **local_interface**를 **em1**로 설정합니다. 설정 스크립트는 이 인터페이스를 **inspection_interface** 매개변수로 정의된 사용자 브릿지에 연결합니다.

local_ip

director 프로비저닝 NIC에 대해 정의된 IP 주소입니다. director에서 DHCP 및 PXE 부팅 서비스에 사용하는 IP 주소이기도 합니다. 프로비저닝 네트워크에 다른 서브넷을 사용하지 않는 경우, 예를 들어 이 IP 주소가 해당 환경의 기존 IP 주소 또는 서브넷과 충돌하는 경우 이 값을 기본값인 **192.168.24.1/24**로 유지하십시오.
IPv6의 경우 상태 저장 및 상태 비저장 연결을 모두 지원하려면 로컬 IP 주소 접두사 길이는 **/64** 여야 합니다.

local_mtu

local_interface에 사용할 MTU(최대 전송 단위)입니다. 언더클라우드의 경우 1500을 초과하지 마십시오.

local_subnet

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

net_config_override

네트워크 구성 덮어쓰기 템플릿의 경로입니다. 이 매개변수를 설정하면 언더클라우드는 JSON 또는 YAML 포멧 템플릿을 사용하여 **os-net-config**로 네트워킹을 구성하고 **undercloud.conf**에 설정된 네

트워크 매개변수를 무시합니다. 본딩을 구성하거나 인터페이스에 옵션을 추가하려는 경우 이 매개변수를 사용합니다. 언더클라우드 네트워크 인터페이스 사용자 지정에 대한 자세한 내용은 언더클라우드 네트워크 인터페이스 구성을 참조하십시오.

networks_file

heat에 대해 오버라이드할 네트워크 파일입니다.

output_dir

상태, 처리된 heat 템플릿, Ansible 배포 파일을 출력할 디렉터리입니다.

overcloud_domain_name

오버클라우드를 배포할 때 사용할 DNS 도메인 이름입니다.

참고

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

roles_file

언더클라우드 설치를 위한 기본 역할 파일을 덮어쓰는 데 사용할 역할 파일입니다. director 설치에 기본 역할 파일이 사용되도록 이 매개변수를 설정되지 않은 상태로 두는 것이 좋습니다.

scheduler_max_attempts

스케줄러가 인스턴스 배포를 시도하는 최대 횟수입니다. 이 값은 스케줄링할 때 잠재적인 경합 조건을 피하기 위해 즉시 배포해야 하는 베어 메탈 노드 수보다 크거나 같아야 합니다.

service_principal

인증서를 사용하는 서비스에 대한 Kerberos 사용자입니다. FreeIPA와 같이 CA에 Kerberos 사용자가 필요한 경우에만 이 매개변수를 사용합니다.

subnets

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

templates

재정의할 heat 템플릿 파일입니다.

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_admin_host 가 **local_ip** 와 동일한 IP 네트워크에 없는 경우 **ControlVirtualInterface** 매개변수를 언더클라우드의 관리자 API를 수신 대기할 인터페이스로 설정해야 합니다. 기본적으로 관리 API는 **br-ctlplane** 인터페이스에서 수신 대기합니다. 사용자 지정 환경 파일에서 **ControlVirtualInterface** 매개변수를 설정하고 **custom_env_files** 매개변수를 구성하여 **undercloud.conf** 파일에 사용자 지정 환경 파일을 포함합니다.

언더클라우드 네트워크 인터페이스 사용자 지정에 대한 자세한 내용은 언더클라우드 네트워크 인터페이스 구성을 참조하십시오.

undercloud_debug

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

undercloud_enable_selinux

배포 중 SELinux를 사용 또는 사용 안 함으로 설정합니다. 문제를 디버깅하지 않는 경우 이 값을 **true**로 설정된 상태로 두는 것이 좋습니다.

undercloud_hostname

언더클라우드에 대해 정규화된 호스트 이름을 정의합니다. 설정되어 있는 경우 언더클라우드 설치 시 모든 시스템 호스트 이름 설정이 구성됩니다. 설정되어 있지 않은 경우 언더클라우드에서 현재 호스트 이름을 사용하지만 사용자가 모든 시스템 호스트 이름 설정을 적절하게 구성해야 합니다.

undercloud_log_file

언더클라우드 설치 및 업그레이드 로그를 저장할 로그 파일 경로입니다. 기본적으로 로그 파일은 홈 디렉터리에 있는 **install-undercloud.log**입니다. 예를 들면 **/home/stack/install-undercloud.log**입니다.

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.

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_public_host 가 **local_ip** 와 동일한 IP 네트워크에 없는 경우 **PublicVirtualInterface** 매개 변수를 언더클라우드의 공용 API를 수신 대기할 공용용 인터페이스로 설정해야 합니다. 기본적으로 공용 API는 **br-ctlplane** 인터페이스에서 수신 대기합니다. 사용자 지정 환경 파일에 **PublicVirtualInterface** 매개변수를 설정하고 **custom_env_files** 매개변수를 구성하여 **undercloud.conf** 파일에 사용자 지정 환경 파일을 포함합니다.

언더클라우드 네트워크 인터페이스 사용자 지정에 대한 자세한 내용은 언더클라우드 네트워크 인터페이스 구성을 참조하십시오.

undercloud_service_certificate

OpenStack SSL/TLS 통신을 위한 인증서 위치 및 파일 이름입니다. 이 인증서를 신뢰할 수 있는 인증 기관에서 가져오는 것이 가장 좋습니다. 또는 자체 서명된 고유 인증서를 생성합니다.

undercloud_timezone

언더클라우드의 호스트 시간대입니다. 시간대를 지정하지 않으면 director는 기존의 표준 시간대 설정을 사용합니다.

undercloud_update_packages

언더클라우드 설치 중 패키지 업데이트 여부를 정의합니다.

서브넷

각 프로비저닝 서브넷은 **undercloud.conf** 파일에서 이름이 지정된 섹션입니다. 예를 들어 **ctlplane-subnet**이라는 서브넷을 생성하려면 **undercloud.conf** 파일에서 다음 샘플을 사용합니다.

```
[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.

중요

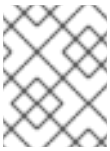
director가 서브넷을 생성한 후에는 director가 서브넷의 IP 주소를 변경할 수 없습니다.

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 프로비저닝 네트워크에서 외부 액세스가 가능합니다.



참고

director 구성은 적절한 **sysctl** 커널 매개변수를 사용하여 IP 포워딩을 자동으로 활성화합니다.

dhcp_start; dhcp_end

오버클라우드 노드의 DHCP 할당 범위 시작과 끝 값입니다. 이 범위에 노드에 할당할 충분한 IP 주소가 포함되어 있는지 확인합니다. 서브넷에 지정되지 않은 경우 director는 서브넷 전체 IP 범위에서 **local_ip**,**게이트웨이**,**undercloud_admin_host**,**undercloud_public_host**, **inspection_iprange** 매개변수에 설정된 값을 제거하여 할당 풀을 결정합니다.
시작 및 종료 주소 쌍 목록을 지정하여 언더클라우드 컨트롤 플레인 서브넷에 대해 일치하지 않는 할당 풀을 구성할 수 있습니다. 또는 **dhcp_exclude** 옵션을 사용하여 IP 주소 범위 내의 IP 주소를 제외할 수 있습니다. 예를 들어 다음 구성에서는 할당 풀 **172.20.0.100-172.20.0.150** 및 **172.20.0.200-172.20.0.250** 을 생성합니다.

옵션 1

```
dhcp_start = 172.20.0.100,172.20.0.200
dhcp_end = 172.20.0.150,172.20.0.250
```

옵션 2

```
dhcp_start = 172.20.0.100
dhcp_end = 172.20.0.250
dhcp_exclude = 172.20.0.151-172.20.0.199
```

dhcp_exclude

DHCP 할당 범위에서 제외할 IP 주소입니다. 예를 들어 다음 구성은 IP 주소 **172.20.0.105** 및 IP 주소 범위 **172.20.0.210-172.20.0.219** 를 제외합니다.

```
dhcp_exclude = 172.20.0.105,172.20.0.210-172.20.0.219
```

dns_nameservers

서브넷과 관련된 DNS 네임 서버입니다. 서브넷의 네임 서버가 정의되지 않은 경우 서브넷에서는 **undercloud_nameservers** 매개변수에 정의된 네임 서버를 사용합니다.

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.

host_routes

이 네트워크의 오버클라우드 인스턴스에 대한 Neutron 관리 서브넷의 호스트 경로입니다. 언더클라우드 **local_subnet**의 호스트 경로도 구성합니다.

inspection_iprange

이 네트워크에서 노드가 검사 프로세스 중에 사용할 임시 IP 범위입니다. 이 범위는 **dhcp_start** 및 **dhcp_end**에 정의된 범위와 겹치지 않아야 하며, 동일한 IP 서브넷에 있어야 합니다.

이러한 매개변수의 값을 설정에 맞게 수정합니다. 완료되면 파일을 저장합니다.

4.3. 환경 파일을 사용하여 언더클라우드 설정

undercloud.conf 파일을 통해 언더클라우드의 기본 매개변수를 구성합니다. heat 매개변수가 포함된 환경 파일을 사용하여 언더클라우드를 추가로 구성할 수도 있습니다.

절차

1. **/home/stack/templates/custom-undercloud-params.yaml**이라는 환경 파일을 생성합니다.
2. 이 파일을 편집하고 heat 매개변수를 포함합니다. 예를 들어 특정 OpenStack Platform 서비스의 디버깅을 활성화하려면 **custom-undercloud-params.yaml** 파일에 다음 스니펫을 포함합니다.

```
parameter_defaults:
  Debug: True
```

   완료되면 이 파일을 저장합니다.

3. **undercloud.conf** 파일을 편집하고 **custom_env_files** 매개변수까지 스크롤합니다. **custom-undercloud-params.yaml** 환경 파일을 가리키도록 매개변수를 편집합니다.

```
custom_env_files = /home/stack/templates/custom-undercloud-params.yaml
```

참고

쉼표로 목록을 구분하여 여러 환경 파일을 지정할 수 있습니다.

다음번 언더클라우드 설치 또는 업그레이드 작업 시 이 환경 파일이 director 설치에 추가됩니다.

4.4. 언더클라우드 구성에 사용되는 일반적인 HEAT 매개변수

다음 표에는 언더클라우드의 사용자 지정 환경 파일에 설정할 수 있는 일반적인 heat 매개변수가 나와 있습니다.

매개변수	설명
AdminPassword	언더클라우드 **admin** 사용자 암호를 설정합니다.

매개변수	설명
AdminEmail	언더클라우드 **admin** 사용자 이메일 주소를 설정합니다.
Debug	디버그 모드를 활성화합니다.

사용자 지정 환경 파일에서 **parameter_defaults** 섹션의 다음 매개변수를 설정합니다.

```
parameter_defaults:
  Debug: True
  AdminPassword: "myp@ssw0rd!"
  AdminEmail: "admin@example.com"
```

4.5. 언더클라우드에서 HIERADATA 구성

director에서 Puppet hieradata를 구성하여 사용 가능한 **undercloud.conf** 매개변수 이외의 사용자 지정 구성을 서비스에 제공할 수 있습니다.

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 다음 스니펫을 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** 파일에서 **hieradata_override** 매개변수를 새 **/home/stack/hieradata.yaml** 파일의 경로로 설정합니다.

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

4.6. IPV6을 통해 베어 메탈 프로비저닝의 언더클라우드 설정

IPv6 노드 및 인프라가 있는 경우 director에서 Red Hat OpenStack Platform을 IPv6 노드에 프로비저닝하고 배포할 수 있도록 IPv4 대신 IPv6을 사용하게 언더클라우드 및 프로비저닝 네트워크를 설정할 수 있습니다. 여기에는 몇 가지 고려 사항이 있습니다.

- Dual stack IPv4/6을 사용할 수 없습니다.
- Tempest 검증이 제대로 수행되지 않을 수 있습니다.
- IPv4에서 IPv6으로 마이그레이션은 업그레이드중에는 사용할 수 없습니다.

undercloud.conf 파일을 수정하여 Red Hat OpenStack Platform에서 IPv6 프로비저닝을 활성화합니다.

사전 요구 사항

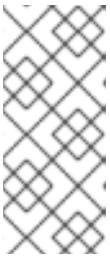
- 언더클라우드의 IPv6 주소. 자세한 내용은 *오버클라우드용 IPv6 네트워킹* 가이드의 언더클라우드에서 IPv6 주소 설정을 참조하십시오.

절차

1. **undercloud.conf** 파일을 엽니다.
2. IPv6 주소 모드를 stateless 또는 stateful으로 지정합니다.

```
[DEFAULT]
ipv6_address_mode = <address_mode>
...
```

NIC **에서 지원하는 모드를 기반으로 <address_mode** >를 **dhcpv6-stateless** 또는 **dhcpv6-stateful** 로 바꿉니다.



참고

상태 저장 주소 모드를 사용하면 펌웨어, 체인 로더 및 운영 체제에서 서로 다른 알고리즘을 사용하여 DHCP 서버가 추적하는 ID를 생성할 수 있습니다. DHCPv6에서는 MAC별 주소를 추적하지 않으며 요청자의 식별자 값이 변경되었지만 MAC 주소는 동일하게 유지됩니다. 따라서 stateful DHCPv6을 사용하는 경우 네트워크 인터페이스를 구성하기 위해 다음 단계를 완료해야 합니다.

3. 상태 저장 DHCPv6을 사용하도록 언더클라우드를 구성한 경우 베어 메탈 노드에 사용할 네트워크 인터페이스를 지정합니다.

```
[DEFAULT]
ipv6_address_mode = dhcpv6-stateful
ironic_enabled_network_interfaces = neutron,flat
...
```

4. 베어 메탈 노드의 기본 네트워크 인터페이스를 설정합니다.

```
[DEFAULT]
...
ironic_default_network_interface = neutron
...
```

5. 언더클라우드에서 provisioning 네트워크에 라우터를 생성할지 여부를 지정합니다.

```
[DEFAULT]
...
enable_routed_networks: <true/false>
...
```

- & **lt;true/false** >를 **true** 로 교체하여 라우팅된 네트워크를 활성화하고 언더클라우드에서 프로비저닝 네트워크에서 라우터를 생성하지 못하도록 합니다. **true** 인 경우 데이터 센터 라우터에서 라우터 알림을 제공해야 합니다.

- 라우팅된 네트워크를 비활성화하려면 &lt **;true/ false** >를 false로 바꾸고 provisioning 네트워크에서 라우터를 생성합니다.

6. SSL/TLS를 통해 로컬 IP 주소와 director Admin API 및 공용 API 끝점의 IP 주소를 구성합니다.

```
[DEFAULT]
...
local_ip = <ipv6_address>
undercloud_admin_host = <ipv6_address>
undercloud_public_host = <ipv6_address>
...
```

& **lt;ipv6_address&** gt;를 언더클라우드의 IPv6 주소로 바꿉니다.

7. 선택 사항: director에서 인스턴스를 관리하는 데 사용하는 프로비저닝 네트워크를 구성합니다.

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
...
```

- 기본 프로비저닝 네트워크를 사용하지 않는 경우 인스턴스 관리에 사용할 < **ipv6_address** > 를 네트워크의 IPv6 주소로 바꿉니다.

- & **lt;ipv6_prefix** >를 기본 프로비저닝 네트워크를 사용하지 않는 경우 인스턴스를 관리하는 데 사용할 네트워크의 IP 주소 접두사로 바꿉니다.

8. 노드 프로비저닝을 위해 DHCP 할당 범위를 구성합니다.

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
...
```

- & **lt;ipv6_address_dhcp_start** >를 오버클라우드 노드에 사용할 네트워크 범위 시작의 IPv6 주소로 바꿉니다.

- & **lt;ipv6_address_dhcp_end** >를 오버클라우드 노드에 사용할 네트워크 범위 끝의 IPv6 주소로 바꿉니다.

9. 선택 사항: 트래픽을 외부 네트워크로 전달할 게이트웨이를 구성합니다.

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
```

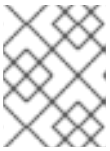
```
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
...
```

기본 **게이트웨이를 사용하지 않는 경우 <ipv6_gateway_address>**를 게이트웨이의 IPv6 주소로 바꿉니다.

10. 검사 프로세스 중에 사용할 DHCP 범위를 구성합니다.

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
...
```

- **검사 프로세스 중에 사용할 네트워크 범위 시작의 IPv6 주소로 <ipv6_**address_inspection_start>를 바꿉니다.
- 검사 프로세스 중에 사용할 네트워크 범위 끝에 < **ipv6_address_inspection_end** >를 IPv6 주소로 바꿉니다.



참고

이 범위는 **dhcp_start** 및 **dhcp_end** 에 정의된 범위와 겹치지 않아야 하지만 동일한 IP 서브넷에 있어야 합니다.

11. 서브넷의 IPv6 이름 서버를 구성합니다.

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
dns_nameservers = <ipv6_dns>
```

& **lt;ipv6_dns&** gt;를 서브넷과 관련된 DNS 이름 서버로 교체합니다.

4.7. 언더클라우드 네트워크 인터페이스 구성

특정 네트워킹 기능을 사용하여 언더클라우드를 설치하려면 **undercloud.conf** 파일에 사용자 지정 네트워크 구성을 포함합니다. 예를 들어 일부 인터페이스에 DHCP가 없을 수 있습니다. 이 경우 **os-net-config**가 언더클라우드 설치 프로세스 중에 구성을 적용할 수 있도록 **undercloud.conf** 파일에서 이러한 인터페이스의 DHCP를 비활성화해야 합니다.

절차

1. 언더클라우드 호스트에 로그인합니다.
2. 새 파일 **undercloud-os-net-config.yaml**을 생성하고 필요한 네트워크 구성을 포함합니다. 자세한 내용은 *Advanced Overcloud Customization* 가이드의 네트워크 인터페이스 참조 를 참조하십시오.

예를 들면 다음과 같습니다.

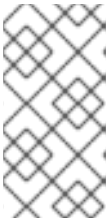
```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

특정 인터페이스에 대한 네트워크 본딩을 생성하려면 다음 샘플을 사용합니다.

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
    - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - name: bond-ctlplane
    type: linux_bond
    use_dhcp: false
    bonding_options: "mode=active-backup"
    mtu: 1500
    members:
    - type: interface
      name: nic2
    - type: interface
      name: nic3
```

3. **undercloud.conf** 파일의 **net_config_override** 매개변수에 **undercloud-os-net-config.yaml** 파일의 경로를 포함합니다.

```
[DEFAULT]
...
net_config_override=undercloud-os-net-config.yaml
...
```



참고

director는 **net_config_override** 매개변수에 포함된 파일을 템플릿으로 사용하여 **/etc/os-net-config/config.yaml** 파일을 생성합니다. **os-net-config** 는 템플릿에 정의된 인터페이스를 관리하므로 이 파일에서 모든 언더클라우드 네트워크 인터페이스 사용자 지정을 수행해야 합니다.

4. 언더클라우드를 설치합니다.

검증

- 언더클라우드 설치가 완료되면 **/etc/os-net-config/config.yaml** 파일에 관련 구성이 포함되어 있는지 확인합니다.

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
    - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

4.8. DIRECTOR 설치

다음 단계에 따라 director를 설치하고 몇 가지 기본적인 설치 후 작업을 수행합니다.

절차

1. 다음 명령을 실행하여 언더클라우드에 director를 설치합니다.

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

이 명령은 director 설정 스크립트를 실행합니다. director가 추가 패키지를 설치하고, **undercloud.conf**의 설정에 따라 해당 서비스를 구성합니다. 이 스크립트를 완료하는 데 몇 분이 걸립니다.

이 스크립트는 다음 두 개의 파일을 생성합니다.

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

2. 이 스크립트는 모든 OpenStack Platform 서비스 컨테이너를 자동으로 시작합니다. 다음 명령을 사용하여 활성화된 컨테이너를 확인할 수 있습니다.

```
[stack@director ~]$ sudo podman ps
```

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

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

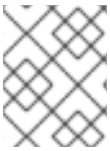
OpenStack 명령이 언더클라우드에 대해 인증되어 실행됨을 나타내는 프롬프트 메시지가 표시됩니다.

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

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

4.9. 오버클라우드의 CPU 아키텍처 구성

RHOSP(Red Hat OpenStack Platform)는 오버클라우드의 CPU 아키텍처를 기본적으로 x86_64로 구성합니다. POWER(ppc64le) 하드웨어에 오버클라우드 컴퓨팅 노드를 배포할 수도 있습니다. 컴퓨팅 노드 클러스터의 경우 동일한 아키텍처를 사용하거나 x86_64 및 ppc64le 시스템을 조합해서 사용할 수 있습니다.



참고

언더클라우드, 컨트롤러 노드, Ceph Storage 노드 및 기타 모든 시스템은 x86_64 하드웨어에서만 지원됩니다.

4.9.1. POWER(ppc64le)를 오버클라우드의 단일 CPU 아키텍처로 구성

오버클라우드에서 컴퓨팅 노드의 기본 CPU 아키텍처는 x86_64입니다. POWER(ppc64le) 하드웨어에 오버클라우드 컴퓨팅 노드를 배포하려면 아키텍처를 ppc64le로 변경할 수 있습니다.

절차

1. **undercloud.conf** 파일에서 iPXE를 비활성화합니다.

```
[DEFAULT]
ipxe_enabled = False
```

참고

RHOSP 16.2.1 및 이전 버전의 경우 이 구성으로 인해 배포의 모든 x86_64 노드가 PXE/legacy 모드로 부팅됩니다. 오버클라우드에 대한 여러 CPU 아키텍처를 구성하려면 여러 CPU 아키텍처 오버클라우드 구성을 참조하십시오.

2. 언더클라우드를 설치합니다.

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

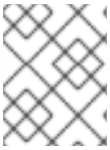
자세한 내용은 언더클라우드에 director 설치를 참조하십시오.

3. 설치 스크립트가 완료될 때까지 기다리십시오.

4. 오버클라우드 노드의 이미지를 가져와서 업로드합니다. 자세한 내용은 Obtaining images for overcloud nodes 에서 참조하십시오.

4.9.2. 여러 CPU 아키텍처 오버클라우드 구성

RHOSP 16.2.2 이상의 경우 아키텍처에 POWER(ppc64le) 및 x86_64 UEFI 노드가 모두 포함된 경우 PXE 및 iPXE 부팅 모드를 모두 지원하도록 언더클라우드를 구성할 수 있습니다.



참고

아키텍처에 POWER(ppc64le) 노드가 포함된 경우 RHOSP 16.2.1 및 이전 버전은 PXE 부팅만 지원합니다.

절차

1. **undercloud.conf** 파일에서 iPXE를 활성화합니다.

```
[DEFAULT]
ipxe_enabled = True
```

2. 언더클라우드의 사용자 지정 환경 파일 **undercloud_noIronicIPXEEnabled.yaml** 을 생성합니다.

3. 기본 Bare Metal Provisioning 서비스(ironic) iPXE 설정을 **PXE** 로 변경하려면 **undercloud_noIronicIPXEEnabled.yaml** 에 다음 설정을 추가합니다.

```
parameter_defaults:
  IronicIPXEEnabled: false
  IronicInspectorIPXEEnabled: true
```

4. 아키텍처에 **ppc64le** 노드가 포함된 경우 **undercloud_noIronicIPXEEnabled.yaml** 에 다음 설정을 추가하여 부팅 제한 시간을 비활성화합니다.

```
parameter_defaults:
  ExtraConfig:
    ironic::config::ironic_config:
      ipmi/disable_boot_timeout:
        value: 'false'
```

5. **undercloud.conf** 파일에 사용자 지정 환경 파일을 추가합니다.

```
[DEFAULT]
...
custom_env_files = undercloud_noIronicIPXEEnabled.yaml
```

6. 언더클라우드를 설치합니다.

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

자세한 내용은 언더클라우드에 director 설치를 참조하십시오.

7. 설치 스크립트가 완료될 때까지 기다리십시오.

8. 오버클라우드 노드를 등록합니다.

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

오버클라우드 노드 등록에 대한 자세한 내용은 오버클라우드 노드 등록을 참조하십시오.

9. 노드 등록 및 구성이 완료될 때까지 기다립니다.
10. director가 노드를 성공적으로 등록했는지 확인합니다.

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

11. 등록된 각 노드의 기존 기능을 확인합니다.

```
$ openstack baremetal node show <node> -f json -c properties | jq -r .properties.capabilities
```

12. 노드의 기존 기능에 **boot_mode: uefi 를 추가하여 각 등록된 노드의 부팅 모드를 uefi로 설정합니다.**

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,<capability_1>,...,
<capability_n>" <node>
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.
- & **lt;capability_1** > 및 모든 기능을 < **capability_n** >까지 바꿉니다.

13. 오버클라우드 노드의 이미지를 가져와서 업로드합니다. 자세한 내용은 여러 CPU 아키텍처 오버클라우드 이미지를 참조하십시오.
14. 각 노드의 부팅 모드를 설정합니다.
    - legacy/PXE의 경우:

```
$ openstack baremetal node set --boot-interface pxe <node_name>
```

    - iPXE의 경우 :

```
$ openstack baremetal node set --boot-interface ipxe <node_name>
```

4.9.3. 다중 아키텍처 오버클라우드에서 Ceph Storage 사용

다중 아키텍처 클라우드에서 외부 Ceph에 대한 액세스 권한을 구성하는 경우 **CephAnsiblePlaybook** 매개변수를 **/usr/share/ceph-ansible/site.yml.sample**로 설정하고 클라이언트 키 및 기타 Ceph 특정 매개변수를 포함합니다.

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
```

4.9.4. 다중 아키텍처 오버클라우드에서 구성 가능 서비스 사용

일반적으로 컨트롤러 노드의 일부인 다음 서비스는 사용자 지정 역할에 기술 프리뷰로 제공되므로 일부는 Red Hat에서 지원하지 않습니다.

- Block Storage 서비스(cinder)
- Image 서비스(glance)
- Identity 서비스(keystone)
- Networking 서비스(neutron)
- Object Storage 서비스(swift)

참고

Red Hat은 기술 프리뷰의 기능을 지원하지 않습니다.

구성 가능 서비스에 관한 자세한 내용은 *Advanced Overcloud Customization* 가이드의 composable services and custom roles를 참조하십시오. 다음 예제를 사용하여 위의 서비스를 컨트롤러 노드에서 전용 ppc64le 노드로 이동하는 방법을 확인합니다.

```
(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
```

4.10. 오버클라우드 노드의 이미지 가져오기

director에는 오버클라우드 노드를 프로비저닝하기 위한 여러 개의 디스크 이미지가 필요합니다.

- PXE 부팅을 통해 베어 메탈 시스템의 인트로스펙션에 사용되는 인트로스펙션 커널 및 램디스크
- 시스템 프로비저닝 및 배포에 사용되는 배포 커널 및 램디스크
- director가 노드의 하드 디스크에 쓰는 기본 오버클라우드 시스템을 생성하는 오버클라우드 커널, 램디스크 및 전체 이미지

CPU 아키텍처에 따라 필요한 이미지를 가져와서 설치할 수 있습니다. 다른 RHOSP(Red Hat OpenStack Platform) 서비스를 실행하지 않으려는 경우 베어 OS를 프로비저닝하기 위해 기본 이미지를 가져와서 설치하거나 서브스크립션 인타이틀먼트 중 하나를 사용할 수도 있습니다.

4.10.1. 단일 CPU 아키텍처 오버클라우드 이미지

RHOSP(Red Hat OpenStack Platform) 설치에는 director용 다음 오버클라우드 이미지를 제공하는 패키지가 포함되어 있습니다.

- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

이러한 이미지는 기본 CPU 아키텍처 x86-64를 사용하여 오버클라우드를 배포하는 데 필요합니다. 이러한 이미지를 director로 가져오면 director PXE 서버에 인트로스펙션 이미지도 설치됩니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

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

3. **rhosp-director-images** 및 **rhosp-director-images-ipa-x86_64** 패키지를 설치합니다.

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images rhosp-director-
images-ipa-x86_64
```

4. **stack** 사용자의 홈 디렉터리(**/home/stack/ images**)에 images 디렉터리를 생성합니다.

```
(undercloud) [stack@director ~]$ mkdir /home/stack/images
```

5. **images** 디렉터리에 압축된 이미지 파일을 풉니다.

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-16.2.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-16.2.tar; do tar -xvf $i; done
```

6. 이미지를 director로 가져옵니다.

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

7. 이미지가 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 |
+--------------------------------------+------------------------+
```

8. director가 인트로스펙션 PXE 이미지를 **/var/lib/ironic/httpboot** 에 복사했는지 확인합니다.

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/httpboot
total 417296
-rwxr-xr-x. 1 root  root    6639920 Jan 29 14:48 agent.kernel
-rw-r--r--. 1 root  root  420656424 Jan 29 14:48 agent.ramdisk
-rw-r--r--. 1 42422 42422       758 Jan 29 14:29 boot.ipxe
-rw-r--r--. 1 42422 42422       488 Jan 29 14:16 inspector.ipxe
```

4.10.2. 여러 CPU 아키텍처 오버클라우드 이미지

RHOSP(Red Hat OpenStack Platform) 설치에는 기본 CPU 아키텍처 x86-64를 사용하여 오버클라우드를 배포하는 데 필요한 다음 이미지를 제공하는 패키지가 포함되어 있습니다.

- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

RHOSP 설치에는 POWER(ppc64le) CPU 아키텍처를 사용하여 오버클라우드를 배포하는 데 필요한 다음 이미지를 제공하는 패키지가 포함되어 있습니다.

- **ppc64le-overcloud-full**

이러한 이미지를 director로 가져오면 director PXE 서버에 인트로스펙션 이미지도 설치됩니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

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

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

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

4. **stack** 사용자의 홈 디렉터리(**/home/stack/images**)에 있는 **images** 디렉터리의 아키텍처별 디렉터리에 압축 파일을 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-16.1-${arch}.tar /usr/share/rhosp-
director-images/ironic-python-agent-latest-16.1-${arch}.tar ; do tar -C $arch -xf $i ; done ;
done
```

5. 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
/var/lib/ironic/tftpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/ppc64le --architecture ppc64le --whole-disk --image-type ironic-python-agent --
http-boot /var/lib/ironic/httpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64/ --architecture x86_64 --http-boot /var/lib/ironic/tftpboot
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64 --architecture x86_64 --image-type ironic-python-agent --http-boot
/var/lib/ironic/httpboot
```

6. 이미지가 업로드되었는지 확인합니다.

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+--------+
| ID                                   | Name                    | Status |
+--------------------------------------+-------------------------+--------+
| 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 |
| f0fedcd0-3f28-4b44-9c88-619419007a03 | ppc64le-overcloud-full  | active |
+--------------------------------------+-------------------------+--------+
```

7. director가 인트로스펙션 PXE 이미지를 **/var/lib/ironic/tftpboot** 에 복사했는지 확인합니다.

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/tftpboot
/var/lib/ironic/tftpboot/ppc64le/
/var/lib/ironic/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 42422 42422      20832 Aug  8 02:08 chain.c32
-rwxr--r--. 1 42422 42422     715584 Aug  8 02:06 ipxe.efi
-rw-r--r--. 1 root  root          22 Aug  8 02:06 map-file
drwxr-xr-x. 2 42422 42422         62 Aug  8 19:34 ppc64le
-rwxr--r--. 1 42422 42422      26826 Aug  8 02:08 pxelinux.0
drwxr-xr-x. 2 42422 42422         21 Aug  8 02:06 pxelinux.cfg
-rwxr--r--. 1 42422 42422      69631 Aug  8 02:06 undionly.kpxe

/var/lib/ironic/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 42422 42422       336 Aug  8 02:06 default
```

4.10.3. 컨테이너 이미지에서 여러 CPU 아키텍처 활성화

RHOSP(Red Hat OpenStack Platform) 배포에 여러 CPU 아키텍처가 있고 컨테이너 이미지를 사용하는 경우 컨테이너 이미지를 업데이트하여 여러 아키텍처를 활성화해야 합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

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

3. **containers-prepare-parameter.yaml** 파일에 추가 아키텍처를 추가하여 여러 아키텍처를 활성화합니다.

```
parameter_defaults:
  ContainerImageRegistryLogin: true
  AdditionalArchitectures: [<list_of_architectures>]
  ContainerImagePrepare:
  - push_destination: true
    ...
```

   & **lt;list_of_architectures** >를 오버클라우드 환경에서 지원하는 아키텍처의 쉼표로 구분된 목록으로 바꿉니다(예: **[ppc64le**]).

4. 컨테이너를 준비하고 업로드합니다.

```
$ openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```

4.10.4. 최소 오버클라우드 이미지

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

RHOSP 설치에는 director의 다음 오버클라우드 이미지를 제공하는 **overcloud-minimal** 패키지가 포함되어 있습니다.

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

참고

기본 **overcloud-full.qcow2** 이미지는 플랫 파티션 이미지입니다. 하지만 전체 디스크 이미지를 가져와서 사용할 수도 있습니다. 자세한 내용은 24장. *전체 디스크 이미지 생성*의 내용을 참조하십시오.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

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

3. **overcloud-minimal** 패키지를 설치합니다.

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

4. **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-16.2.tar
```

5. 이미지를 director로 가져옵니다.

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

6. 이미지가 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 |
+--------------------------------------+-------------------------+
```

4.11. 컨트롤 플레인의 네임서버 설정

오버클라우드에서 **cdn.redhat.com**과 같은 외부 호스트 이름을 확인하도록 하려면 오버클라우드 노드에 네임서버를 설정합니다. 네트워크를 분리하지 않은 표준 오버클라우드의 경우 언더클라우드 컨트롤 플레인 서브넷을 사용하여 네임서버를 정의합니다. 다음 절차에 따라 해당 환경의 네임서버를 정의하십시오.

절차

1. **stackrc** 파일을 소싱하여 director 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 사용해야 합니다. 컨트롤 플레인 이름 서버와 **DnsServers** 매개변수도 동일한 DNS 서버로 설정해야 합니다.

4.12. 언더클라우드 설정 업데이트

새로운 요구 사항에 맞게 언더클라우드 설정을 변경해야 하는 경우 설치 후 언더클라우드 설정을 변경하고, 관련 설정 파일을 편집한 후, **openstack undercloud install** 명령을 다시 실행하면 됩니다.

절차

1. 언더클라우드 설정 파일을 수정합니다. 예를 들어 **undercloud.conf** 파일을 편집하고 사용 가능한 하드웨어 유형 목록에 **idrac** 하드웨어 유형을 추가합니다.

```
enabled_hardware_types = ipmi,redfish,idrac
```

2. **openstack undercloud install** 명령을 실행하여 새로운 변경 사항으로 언더클라우드를 새로 고침합니다.

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

   명령이 완료될 때까지 기다립니다.

3. 명령행 툴을 사용하도록 **stack** 사용자를 초기화합니다.

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

이제 OpenStack 명령이 언더클라우드에 대해 인증되어 실행됨을 나타내는 프롬프트 메시지가 표시됩니다.

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

4. director가 새 설정을 적용했는지 확인합니다. 이 예제에서는 활성화된 하드웨어 유형 목록을 확인합니다.

```
(undercloud) [stack@director ~]$ openstack baremetal driver list
+---------------------+----------------------+
| Supported driver(s) | Active host(s)       |
+---------------------+----------------------+
| idrac               | director.example.com |
| ipmi                | director.example.com |
| redfish             | director.example.com |
+---------------------+----------------------+
```

언더클라우드 재설정이 완료되었습니다.

4.13. 언더클라우드 컨테이너 레지스트리

Red Hat Enterprise Linux 8.4에는 Docker Registry v2를 설치한 **docker-distribution** 패키지가 더 이상 포함되지 않습니다. 호환성과 동일한 수준의 기능을 유지하기 위해 director 설치에서 **image-serve**라는 vhost로 Apache 웹 서버를 생성하여 레지스트리를 제공합니다. 이 레지스트리는 SSL이 비활성화된 포트 8787/TCP도 사용합니다. Apache 기반 레지스트리는 컨테이너화되지 않으므로, 다음 명령을 실행하여 레지스트리를 다시 시작해야 합니다.

```
$ sudo systemctl restart httpd
```

컨테이너 레지스트리 로그는 다음 위치에서 찾을 수 있습니다.

- /var/log/httpd/image_serve_access.log
- /var/log/httpd/image_serve_error.log.

이미지 내용은 **/var/lib/image-serve**에서 제공됩니다. 이 위치에서는 특정 디렉터리 레이아웃 및 **apache** 구성을 사용하여 레지스트리 REST API의 풀 기능을 구현합니다.

Apache 기반 레지스트리는 **podman push** 및 **buildah push** 명령을 지원하지 않으므로 기존 방법으로는 컨테이너 이미지를 푸시할 수 없습니다. 배포 중에 이미지를 수정하려면 **ContainerImagePrepare** 매개변수와 같은 컨테이너 준비 워크플로우를 사용합니다. 컨테이너 이미지를 관리하려면 다음과 같은 컨테이너 관리 명령을 사용하십시오.

OpenStack tripleo 컨테이너 이미지 목록

레지스트리에 저장된 모든 이미지를 나열합니다.

OpenStack tripleo 컨테이너 이미지 표시

레지스트리에 있는 특정 이미지의 메타데이터를 표시합니다.

OpenStack tripleo 컨테이너 이미지 푸시

원격 레지스트리에서 언더클라우드 레지스트리로 이미지를 푸시합니다.

OpenStack tripleo container image delete

레지스트리에서 이미지를 삭제합니다.

5장. 언더클라우드 MINION 설치

언더클라우드 minion을 추가로 배포하여 OpenStack Platform director 서비스를 여러 호스트에 확장할 수 있으므로 대규모 오버클라우드를 배포할 때 성능이 향상됩니다. 이 기능은 선택 사항입니다.

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.

5.1. 언더클라우드 MINION

언더클라우드 minion은 별도의 호스트에서 **heat-engine** 및 **ironic-conductor** 서비스를 추가로 제공합니다. 이러한 추가 서비스는 오케스트레이션 및 프로비저닝 작업에서 언더클라우드를 지원합니다. 여러 호스트에 언더클라우드 작업을 배포하면 오버클라우드 배포를 실행하는 데 더 많은 리소스가 제공되어 대규모 배포가 더 신속하게 수행됩니다.

5.2. 언더클라우드 MINION 요구 사항

서비스 요구 사항입니다

언더클라우드 minion의 확장된 **heat-engine** 및 **ironic-conductor** 서비스에서는 작업자 세트를 사용합니다. 각 작업자는 해당 서비스와 관련된 작업을 수행합니다. 여러 작업자를 사용하면 동시에 작업을 수행할 수 있습니다. minion의 기본 작업자 수는 minion 호스트의 전체 CPU 스레드 수를 반으로 나눈 값입니다. 이 경우 전체 스레드 수는 CPU 코어 수에 하이퍼 스레딩 값을 곱한 값입니다. 예를 들어 minion에 16개의 스레드가 있는 CPU가 있으면 minion은 기본적으로 서비스당 8개의 작업자를 생성합니다. 또한 minion에서는 기본적으로 최소 및 최대 한도 세트를 사용합니다.

서비스	최소	최대
heat-engine	4	24
ironic-conductor	2	12

언더클라우드 minion에는 다음과 같은 최소 CPU 및 메모리 요구 사항이 있습니다.

- Intel 64 또는 AMD64 CPU 확장 기능을 지원하는 8스레드 64비트 x86 프로세서. 이 프로세서는 언더클라우드 서비스당 작업자 4개를 제공합니다.
- 최소 16GB의 RAM

다수의 작업자를 사용하려면 CPU 스레드당 2GB의 RAM으로 언더클라우드의 vCPU 및 메모리 수를 늘리십시오. 예를 들어 스레드가 48개인 머신에는 96GB RAM이 있어야 합니다. 24개의 **heat-engine** 작업자 및 12개의 **ironic-conductor** 작업자를 대상으로 합니다.

컨테이너 이미지 요구 사항입니다

언더클라우드 minion은 내부 컨테이너 이미지 레지스트리를 호스팅하지 않습니다. 따라서 다음 방법 중 하나를 사용하여 컨테이너 이미지를 가져오도록 minion을 구성해야 합니다.

- Red Hat Container Image Registry(**registry.redhat.io**)에서 이미지를 직접 가져옵니다.

- Red Hat Satellite Server에서 호스팅하는 이미지를 가져옵니다.

두 방법 모두 **containers-prepare-parameter.yaml** 파일에서 **ContainerImagePrepare** heat 매개변수의 일부로 **push_destination: false**를 설정해야 합니다.

5.3. MINION 준비

minion을 설치하려면 먼저 호스트 머신에서 몇 가지 기본 설정을 완료해야 합니다.

- 명령을 실행할 root가 아닌 사용자
- 확인 가능한 호스트 이름
- Red Hat 서브스크립션
- 이미지 준비 및 minion 설치에 필요한 명령행 툴

절차

1. **root** 사용자로 minion 호스트에 로그인합니다.
2. **stack** 사용자를 생성합니다.

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

3. **stack** 사용자의 암호를 설정합니다.

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

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

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

5. 새로 만든 **stack** 사용자로 전환합니다.

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

6. minion의 기본 및 전체 호스트 이름을 확인합니다.

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

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

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

7. **/etc/hosts** 파일을 편집하고 시스템 호스트 이름을 입력합니다. 예를 들어 시스템 이름이 **minion.example.com**이고 IP 주소 **10.0.0.1**을 사용하는 경우 **/etc/hosts** 파일에 다음 행을 추가합니다.

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

8. Red Hat Content Delivery Network 또는 Red Hat Satellite에 시스템을 등록합니다. 예를 들어 다음 명령을 실행하여 시스템을 콘텐츠 전송 네트워크에 등록합니다. 메시지가 표시되면 고객 포털 사용자 이름과 암호를 입력합니다.

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

9. RHOSP(Red Hat OpenStack Platform) director의 인타이틀먼트 풀 ID를 검색합니다.

```
[stack@minion ~]$ 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
```

10. **Pool ID** 값을 찾아서 Red Hat OpenStack Platform 16.2 인타이틀먼트를 연결합니다.

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

11. 기본 리포지토리를 모두 비활성화하고 필수 Red Hat Enterprise Linux 리포지토리를 활성화합니다.

```
[stack@minion ~]$ sudo subscription-manager repos --disable=*
[stack@minion ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-
eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-
highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-
16.2-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms
```

이러한 리포지토리에는 minion 설치에 필요한 패키지가 들어 있습니다.

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

```
[stack@minion ~]$ sudo dnf update -y
[stack@minion ~]$ sudo reboot
```

13. minion 설치 및 설정에 필요한 명령행 툴을 설치합니다.

```
[stack@minion ~]$ sudo dnf install -y python3-tripleoclient
```

5.4. 언더클라우드 설정 파일을 MINION에 복사

minion 설치 시 minion 서비스를 설정하고 이를 director에 등록하려면 minion에 언더클라우드의 설정 파일이 있어야 합니다.

- **tripleo-undercloud-outputs.yaml**
- **tripleo-undercloud-passwords.yaml**

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. 다음과 같이 언더클라우드에서 minion으로 파일을 복사합니다.

```
$ scp ~/tripleo-undercloud-outputs.yaml ~/tripleo-undercloud-passwords.yaml
stack@<minion-host>:~/.
```

   - **<minion-host>**를 minion의 호스트 이름 또는 IP 주소로 바꿉니다.

5.5. 언더 클라우드 인증 기관 복사

언더클라우드에서 끝점 암호화에 SSL/TLS를 사용하면 minion 호스트에 언더클라우드 SSL/TLS 인증서에 서명한 인증서가 포함되어 있어야 합니다. 언더클라우드 구성에 따라 이 인증 기관은 다음 중 하나입니다.

- 인증서가 minion 호스트에 사전 로드된 외부 인증 기관. 작업이 필요하지 않습니다.
- director가 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem**에서 생성한 director 생성 자체 서명 인증 기관. 이 파일을 minion 호스트에 복사하고 minion 호스트에 대해 신뢰할 수 있는 인증 기관의 일부로 파일을 사용하십시오. 이 절차에서는 이 파일을 예제로 사용합니다.
- OpenSSL로 생성한 사용자 지정 자체 서명 인증 기관. 이 문서의 예제에서는 이 파일을 **ca.crt.pem**으로 참조합니다. 이 파일을 minion 호스트에 복사하고 minion 호스트에 대해 신뢰할 수 있는 인증 기관의 일부로 파일을 사용하십시오.

절차

1. **root** 사용자로 minion 호스트에 로그인합니다.
2. 인증 기관 파일을 언더클라우드에서 minion으로 복사하십시오.

```
[root@minion ~]# scp \
  root@<undercloud-host>:/etc/pki/ca-trust/source/anchors/cm-local-ca.pem \
  /etc/pki/ca-trust/source/anchors/undercloud-ca.pem
```

   - **<undercloud-host>**를 언더클라우드의 호스트 이름 또는 IP 주소로 바꿉니다.

3. minion 호스트의 신뢰할 수 있는 인증 기관을 업데이트하십시오.

```
[root@minion ~]# update-ca-trust enable
[root@minion ~]# update-ca-trust extract
```

-

5.6. MINION 구성

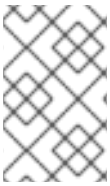
minion 설치 프로세스에는 minion이 **stack** 사용자의 홈 디렉터리에서 읽어오는 **minion.conf** 구성 파일의 특정 설정이 필요합니다. 기본 템플릿을 기본으로 사용하여 구성하려면 다음 단계를 완료합니다.

절차

1. **stack** 사용자로 minion 호스트에 로그인합니다.
2. 기본 템플릿을 **stack** 사용자의 홈 디렉터리에 복사합니다.

```
[stack@minion ~]$ cp \
 /usr/share/python-tripleoclient/minion.conf.sample \
 ~/minion.conf
```

3. **minion.conf** 파일을 편집합니다. 이 파일에는 minion을 구성하는 설정이 포함되어 있습니다. 매개변수를 생략하거나 주석으로 처리하면 minion 설치에 기본값이 사용됩니다. 다음 권장 매개변수를 확인하십시오.
   - **minion_hostname** - minion의 호스트 이름으로 설정합니다.
   - **minion_local_interface** - 프로비저닝 네트워크를 통해 언더클라우드에 연결하는 인터페이스로 설정합니다.
   - **minion_local_ip** - 프로비저닝 네트워크에서 사용 가능한 IP 주소로 설정합니다.
   - **minion_nameservers** - minion이 호스트 이름을 확인할 수 있도록 DNS 네임서버로 설정합니다.
   - **enable_ironic_conductor** - **ironic-conductor** 서비스의 활성화 여부를 정의합니다.
   - **enable_heat_engine** - **heat-engine** 서비스 활성화 여부를 정의합니다.



참고

기본 **minion.conf** 파일을 사용하면 minion에서 **heat-engine** 서비스만 활성화합니다. **ironic-conductor** 서비스를 활성화하려면 **enable_ironic_conductor** 매개변수를 **true**로 설정하십시오.

5.7. MINION 구성 매개변수

다음 목록에는 **minion.conf** 파일을 구성하기 위한 매개변수 정보가 나와 있습니다. 오류를 방지하려면 모든 매개변수를 관련 섹션에 보관합니다.

기본값

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

cleanup

임시 파일을 정리합니다. 명령을 실행한 후 배포 중 사용한 임시 파일을 그대로 두려면 이 매개 변수를 **False** 로 설정합니다. 이 매개변수는 생성된 파일을 디버깅하거나 오류가 발생한 경우에 유용합니다.

container_cli

컨테이너 관리를 위한 CLI 툴입니다. 이 매개변수를 **podman**으로 설정해 둡니다. Red Hat Enterprise Linux 8.4는 **podman**만 지원합니다.

container_healthcheck_disabled

컨테이너화된 서비스 상태 점검을 비활성화합니다. 상태 점검을 활성화하고 이 옵션을 **false**로 설정해 두는 것이 좋습니다.

container_images_file

컨테이너 이미지 정보가 포함된 heat 환경 파일입니다. 이 파일은 다음 항목을 포함할 수 있습니다.

- 필요한 모든 컨테이너 이미지에 대한 매개변수
- 필요한 이미지 준비를 수행하는 **ContainerImagePrepare** 매개변수. 일반적으로 이 매개변수가 포함된 파일의 이름은 **containers-prepare-parameter.yaml**입니다.

container_insecure_registries

podman이 사용할 수 있는 비보안 레지스트리 목록입니다. 개인 컨테이너 레지스트리와 같은 다른 소스에서 이미지를 가져오려는 경우 이 매개변수를 사용합니다. 대부분의 경우 minion이 Satellite에 등록되어 있으면, Red Hat Container Catalog 또는 Satellite 서버에서 컨테이너 이미지를 가져올 수 있는 인증서가 **podman**에 있습니다.

container_registry_mirror

podman에서 사용하도록 구성된 **registry-mirror**(선택 사항)입니다.

custom_env_files

minion 설치에 추가할 추가 환경 파일입니다.

deployment_user

minion을 설치하는 사용자입니다. 현재 기본 사용자인 **stack**을 사용하려면 이 매개변수를 설정되지 않은 상태로 두십시오.

enable_heat_engine

minion에 heat 엔진을 설치할지 여부를 정의합니다. 기본값은 **true**입니다.

enable_ironic_conductor

ironic conductor 서비스를 minion에 설치할지 여부를 정의합니다. 기본값은 **false**입니다. ironic conductor 서비스를 활성화하려면 이 값을 **true**로 설정하십시오.

heat_container_image

사용하려는 heat 컨테이너 이미지의 URL입니다. 설정되지 않은 상태로 두십시오.

heat_native

네이티브 Heat 템플릿을 사용합니다. **true**로 두십시오.

hieradata_override

director에서 Puppet hieradata를 구성하여 **minion.conf** 매개변수 이외의 사용자 지정 설정을 서비스에 제공하는 **hieradata** 오버라이드 파일의 경로입니다. 설정한 경우 minion 설치 시 이 파일이 **/etc/puppet/hieradata** 디렉터리에 복사되고 계층에서 첫 번째 파일로 설정됩니다.

minion_debug

이 값을 **true**로 설정하여 minion 서비스의 **DEBUG** 로그 레벨을 활성화하십시오.

minion_enable_selinux

배포 중 SELinux를 사용 또는 사용 안 함으로 설정합니다. 문제를 디버깅하지 않는 경우 이 값을 **true**로 설정된 상태로 두는 것이 좋습니다.

minion_enable_validations

minion에서 인증 서비스를 활성화합니다.

minion_hostname

minion에 대해 정규화된 호스트 이름을 정의합니다. 설정되어 있는 경우 minion 설치 시 모든 시스템의 호스트 이름이 구성됩니다. 설정되어 있지 않은 경우 minion에서 현재 호스트 이름을 사용하지만 사용자가 모든 시스템 호스트 이름 설정을 적절하게 구성해야 합니다.

minion_local_interface

언더클라우드에서 프로비저닝 NIC용으로 선택한 인터페이스입니다. minion에서 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.

minion_local_ip

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

minion_local_mtu

local_interface에 사용할 MTU(최대 전송 단위)입니다. minion의 경우 1500을 초과하지 마십시오.

minion_log_file

minion 설치 및 업그레이드 로그를 저장할 로그 파일 경로입니다. 기본적으로 로그 파일은 홈 디렉터리에 있는 **install-minion.log**입니다. 예를 들면 **/home/stack/install-minion.log**입니다.

minion_nameservers

minion 호스트 이름 확인에 사용할 DNS 이름 서버 목록입니다.

minion_ntp_servers

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

minion_password_file

minion이 언더클라우드 서비스에 연결하는 데 사용할 암호가 포함된 파일입니다. 이 매개변수는 언더클라우드에서 복사한 **tripleo-undercloud-passwords.yaml** 파일로 설정된 상태로 둡니다.

minion_service_certificate

OpenStack SSL/TLS 통신을 위한 인증서 위치 및 파일 이름입니다. 이 인증서를 신뢰할 수 있는 인증 기관에서 가져오는 것이 가장 좋습니다. 또는 자체 서명된 고유 인증서를 생성합니다.

minion_timezone

minion의 호스트 시간대입니다. 시간대를 지정하지 않으면 minion은 기존의 표준 시간대 설정을 사용합니다.

minion_undercloud_output_file

minion이 언더클라우드 서비스에 연결하는 데 사용할 수 있는 언더클라우드 구성 정보가 포함된 파일입니다. 이 매개변수를 언더클라우드에서 복사한 **tripleo-undercloud-outputs.yaml** 파일로 설정한 상태로 둡니다.

net_config_override

네트워크 구성 덮어쓰기 템플릿의 경로입니다. 이 매개변수를 설정하면 minion은 JSON 포멧 템플릿을 사용하여 **os-net-config**로 네트워킹을 구성하고 **minion.conf**에 설정된 네트워크 매개변수를 무시합니다. 예를 보려면 **/usr/share/python-tripleoclient/minion.conf.sample**을 참조하십시오.

networks_file

heat에 대해 오버라이드할 네트워크 파일입니다.

output_dir

상태, 처리된 heat 템플릿, Ansible 배포 파일을 출력할 디렉터리입니다.

roles_file

minion 설치를 위한 기본 역할 파일을 덮어쓸 역할 파일입니다. minion 설치에 기본 역할 파일이 사용되도록 이 매개변수를 설정되지 않은 상태로 두는 것이 좋습니다.

templates

재정의할 heat 템플릿 파일입니다.

5.8. MINION 설치

minion을 설치하려면 다음 단계를 완료하십시오.

절차

1. **stack** 사용자로 minion 호스트에 로그인합니다.
2. minion을 설치하려면 다음 명령을 실행합니다.

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

이 명령은 minion의 설정 스크립트를 시작하고, 추가 패키지를 설치하며, **minion.conf** 파일의 설정에 따라 minion 서비스를 구성합니다. 이 스크립트를 완료하는 데 몇 분이 걸립니다.

5.9. MINION 설치 확인

성공적으로 minion이 설치되었는지 확인하려면 다음 단계를 완료하십시오.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

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

3. minion에서 heat engine 서비스를 활성화한 경우, minion의 **heat-engine** 서비스가 언더클라우드 서비스 목록에 표시되는지 확인합니다.

```
[stack@director ~]$ $ openstack orchestration service list
```

이 명령 출력에서는 언더클라우드와 minion 모두에 **heat-engine** 작업자가 있는 테이블이 표시됩니다.

4. minion에서 ironic conductor 서비스를 활성화한 경우, minion의 **ironic-conductor** 서비스가 언더클라우드 서비스 목록에 표시되는지 확인합니다.

```
[stack@director ~]$ $ openstack baremetal conductor list
```

이 명령 출력에서는 언더클라우드와 minion 모두에 **ironic-conductor** 서비스가 있는 테이블을 표시합니다.

6장. 오버클라우드 계획

다음 섹션에는 RHOSP(Red Hat OpenStack Platform) 환경의 여러 측면을 계획하기 위한 몇 가지 지침이 나와 있습니다. 여기에는 노드 역할 정의, 네트워크 토폴로지 계획 및 스토리지가 포함됩니다.

중요

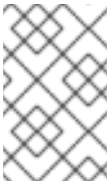
배포 후 오버클라우드 노드의 이름을 바꾸지 마십시오. 배포 후 노드 이름을 변경하면 인스턴스 관리 문제가 발생합니다.

6.1. 노드 역할

director에는 오버클라우드를 빌드하기 위한 다음과 같은 기본 노드 유형이 있습니다.

컨트롤러

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



참고

하나의 컨트롤러 노드가 있는 환경은 프로덕션이 아닌 테스트 목적으로만 사용합니다. 두 개의 컨트롤러 노드 또는 세 개 이상의 컨트롤러 노드로 구성된 환경은 지원되지 않습니다.

컴퓨팅

환경에서 가상 머신을 실행하는 데 필요한 처리 기능이 포함되어 있고 하이퍼바이저 역할을 하는 물리 서버입니다. 기본 RHOSP 환경에는 하나 이상의 컴퓨팅 노드가 필요합니다.

Ceph Storage

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

Swift Storage

OpenStack Object Storage(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 6.1. 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.

표 6.2. 개념 배포 증명을 위한 노드 배포 역할

	언더클라우드	컨트롤러	컴퓨팅	Ceph Storage	합계
개념 증명	1	1	1	1	4

주의

Red Hat OpenStack Platform은 2일차(Day-2) 작업 중에 작동 중인 Ceph Storage 클러스터를 유지 관리합니다. 따라서 Ceph Storage 클러스터의 업그레이드 또는 마이너 업데이트와 같은 일부 2일차 작업은 MON 또는 스토리지 노드가 3개 미만인 배포에서는 수행할 수 없습니다. 단일 Controller 노드 또는 단일 Ceph Storage 노드를 사용하는 경우 2일차 작업이 실패합니다.

6.2. 오버클라우드 네트워크

역할과 서비스가 서로 올바르게 통신할 수 있도록 매핑하려면 해당 환경의 네트워킹 토폴로지와 서브넷을 계획하는 것이 중요합니다. RHOSP(Red Hat OpenStack Platform)에서는 자율적으로 작동하고 소프트웨어 기반 네트워크, 정적 및 유동 IP 주소, DHCP를 관리하는 Openstack Networking(neutron) 서비스를 사용합니다.

기본적으로 director는 연결에 **프로비저닝/컨트롤 플레인**을 사용하도록 노드를 구성합니다. 그러나 사용자 지정하여 서비스를 할당할 수 있는 일련의 구성 가능 네트워크로 네트워크 트래픽을 분리할 수 있습니다.

일반적인 RHOSP 설치에서는 종종 네트워크 유형의 수가 물리적 네트워크 연결 수를 초과합니다. 모든 네트워크를 적절한 호스트에 연결하기 위해 오버클라우드에서는 VLAN 태그를 사용하여 각 인터페이스에서 두 개 이상의 네트워크를 제공합니다. 대부분의 네트워크는 서브넷이 분리되어 있지만, 일부 네트워크는 인터넷 액세스 또는 인프라 네트워크 연결을 위해 라우팅할 레이어 3 게이트웨이가 필요합니다. VLAN을 사용하여 네트워크 트래픽 유형을 분리하는 경우 802.1Q 표준의 스위치를 사용하여 태그된 VLAN을 제공해야 합니다.

참고

배포 시 터널링이 비활성화된 neutron VLAN 모드를 사용하려는 경우에도 프로젝트 네트워크(GRE 또는 VXLAN으로 터널링됨)를 배포하는 것이 좋습니다. 이 경우 배포 시 약간의 사용자 지정 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.

director에는 분리된 구성 가능 네트워크를 사용하여 NIC를 구성하는 데 사용할 수 있는 템플릿 세트가 포함되어 있습니다. 기본 구성은 다음과 같습니다.

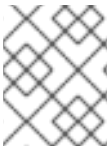
- 단일 NIC 구성 - 기본 VLAN과 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN에서 프로비저닝 네트워크용 NIC 1개
- 본딩된 NIC 구성 - 기본 VLAN의 프로비저닝 네트워크용 NIC 1개와 다른 오버클라우드 네트워크 유형에 대해 태그된 VLAN용 본딩의 NIC 2개
- 다중 NIC 구성 - 각 NIC에서 다른 오버클라우드 네트워크 유형에 서브넷 사용

고유의 템플릿을 작성하여 특정 NIC 구성을 매핑할 수도 있습니다.

네트워크 구성을 검토할 때 다음과 같은 세부 정보도 고려해야 합니다.

- 오버클라우드 생성 중에 모든 오버클라우드 머신에 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 NIC로 PXE를 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.
- director가 각 노드의 전원 관리를 제어하려면 모든 오버클라우드 베어 메탈 시스템에 IPMI(Intelligent Platform Management Interface)와 같은 지원되는 전원 관리 인터페이스가 있어야 합니다.
- 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 액세스해야 하는 경우 공용 네트워크에서 유동 IP 주소를 할당하고 유동 IP를 인스턴스와 연결할 수 있습니다. 이 인스턴스는 해당 개인 IP를 보존하지만, 네트워크 트래픽은 NAT를 사용하여 유동 IP 주소로 이동합니다. 유동 IP 주소는 여러 개인 IP 주소가 아니라 단일 인스턴스에만 할당할 수 있습니다. 하지만 유동 IP 주소는 단일 테넌트에서만 사용하도록 예약됩니다. 즉, 테넌트가 필요에 따라 유동 IP 주소를 특정 인스턴스와 연결하거나 연결 해제할 수 있습니다. 이 설정에서는 인프라가 외부 인터넷에 공개되므로 적절한 보안 사례를 따라야 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.

- 프로비저닝 인터페이스, 외부 인터페이스 및 유동 IP 인터페이스를 1500의 기본 MTU에 두는 것이 좋습니다. 그렇지 않으면 연결 문제가 발생할 수 있습니다. 라우터는 일반적으로 계층 3 경계 간에 점보 프레임을 전달할 수 없기 때문입니다.



참고

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

6.3. 오버클라우드 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. LVM 필터를 사용하여 가시성, 가용성 및 데이터 손상 문제를 완화합니다. 자세한 내용은 *Storage Guide*의 섹션 2 Block Storage and Volumes 및 KCS 문서 3213311, "Using LVM on a cinder volume exposes the data to the compute host"를 참조하십시오.

director에는 오버클라우드 환경에 대한 여러 스토리지 옵션이 포함되어 있습니다.

Ceph Storage 노드

director는 Red Hat Ceph Storage를 사용하여 확장 가능한 스토리지 노드 세트를 생성합니다. 오버클라우드는 이러한 노드를 다음 스토리지 유형에 사용합니다.

- **이미지** - Image 서비스(glance)는 가상 머신의 이미지를 관리합니다. 이미지는 변경할 수 없습니다. OpenStack에서는 이미지를 바이너리 Blob으로 처리하고 그에 따라 이미지를 다운로드합니다. Image 서비스(glance)를 사용하여 이미지를 Ceph 블록 장치에 저장할 수 있습니다.

- **볼륨** - OpenStack은 Block Storage 서비스(cinder)를 사용하여 볼륨을 관리합니다. Block Storage 서비스(cinder) 볼륨은 블록 장치입니다. OpenStack은 볼륨을 사용하여 가상 머신을 부팅하거나 실행 중인 가상 머신에 볼륨을 연결합니다. Block Storage 서비스를 사용하여 이미지의 COW(Copy-On-Write) 복제본을 사용하여 가상 머신을 부팅할 수 있습니다.

- **파일 시스템** - Openstack은 Shared File Systems 서비스(manila)로 공유 파일 시스템을 관리합니다. 공유는 파일 시스템에서 지원됩니다. manila를 사용하면 Ceph Storage 노드의 데이터로 CephFS 파일 시스템에서 지원하는 공유를 관리할 수 있습니다.

- **게스트 디스크** - 게스트 디스크는 게스트 운영 체제 디스크입니다. 기본적으로 Compute 서비스(nova)로 가상 머신을 부팅하면 가상 머신 디스크가 하이퍼바이저의 파일 시스템에 파일로 표시됩니다(일반적으로 **/var/lib/nova/instances/<uuid>**/아래 표시). Ceph 내부의 모든 가상 머신은 Block Storage 서비스(cinder)를 사용하지 않고 부팅할 수 있습니다. 따라서 실시간 마이그레이션 프로세스를 통해 쉽게 유지보수 작업을 수행할 수 있습니다. 또한 하이퍼바이저에 장애가 발생하는 경우 **nova evacuate**를 트리거하고 가상 머신을 다른 위치에서 실행할 수 있으므로 편리합니다.

중요

지원되는 이미지 포맷에 관한 자세한 내용은 *Creating and Managing Images* 가이드의 Image Service 장을 참조하십시오.

Ceph Storage에 관한 자세한 내용은 Red Hat Ceph Storage Architecture Guide 를 참조하십시오.

Swift Storage 노드

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

6.4. 오버클라우드 보안

OpenStack Platform 구현의 보안 수준은 해당 환경의 보안 수준에 따라 좌우됩니다. 해당 네트워킹 환경에 이상적인 보안 원칙에 따라 네트워크 액세스를 적절히 제어해야 합니다.

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

시스템 보안에 대한 자세한 내용은 다음 Red Hat 가이드를 참조하십시오.

- Red Hat Enterprise Linux 8용 Security Hardening
- Red Hat Enterprise Linux 8용 Using SELinux

6.5. 오버클라우드 고가용성

고가용성 오버클라우드를 배포하기 위해 director는 여러 개의 컨트롤러 노드, 컴퓨팅 노드, 스토리지 노드가 단일 클러스터로 작동하도록 구성합니다. 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.

참고

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 Compute 인스턴스의 고가용성을 구성할 수도 있습니다. 이 고가용성 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 HA 및 설치 지침에 관한 자세한 내용은 High Availability for Compute Instances 가이드를 참조하십시오.

6.6. 컨트롤러 노드 요구 사항

컨트롤러 노드는 Red Hat OpenStack Platform 환경에서 대시보드(horizon), 백엔드 데이터베이스 서버, Identity 서비스(keystone) 인증, 고가용성 서비스와 같은 코어 서비스를 호스트합니다.

프로세서

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

메모리

최소 메모리 용량은 32GB입니다. 하지만 권장 메모리 용량은 CPU 코어 수에 하이퍼 스레딩 값을 곱하여 계산된 vCPU 수에 따라 달라집니다. 다음과 같이 계산하여 RAM 요구 사항을 확인하십시오.

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

메모리 요구 사항을 측정하는 방법에 대한 자세한 내용은 Red Hat 고객 포털의 "Red Hat OpenStack Platform Hardware Requirements for Highly Available Controllers"를 참조하십시오.

디스크 스토리지 및 레이아웃

컨트롤러 노드에서 Object Storage 서비스(swift)를 실행하지 않는 경우 최소 50GB의 스토리지가 필요합니다. Telemetry 및 Object Storage 서비스는 모두 컨트롤러에 설치되고 root 디스크를 사용하도록 구성됩니다. 이러한 기본값은 상용 하드웨어에 구축된 소형 오버클라우드를 배포할 때 적합합니다. 이러한 환경은 일반적인 개념 검증 및 테스트 환경입니다. 워크로드 용량 및 성능 면에서는 떨어지지만, 이러한 기본값을 사용하여 최소한의 계획으로 오버클라우드를 배포할 수 있습니다.
하지만 엔터프라이즈 환경에서는 Telemetry가 스토리지에 지속적으로 액세스하므로 기본값을 사용할 경우 심각한 병목 현상이 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 사용하십시오.

전원 관리

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

가상화 지원

Red Hat은 Red Hat Virtualization 플랫폼에서 가상화된 컨트롤러 노드만 지원합니다. 자세한 정보는 가상화된 컨트롤 플레인 생성 을 참조하십시오.

6.7. 컴퓨팅 노드 요구 사항

컴퓨팅 노드는 가상 머신 인스턴스가 시작된 후 이를 실행하는 역할을 합니다. 컴퓨팅 노드에는 하드웨어 가상화를 지원하는 베어 메탈 시스템이 필요합니다. 또한 호스트하는 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 있는 것이 좋습니다.
- IBM POWER 8 프로세서

메모리

호스트 운영 체제용 최소 6GB의 RAM과 다음과 같은 고려 사항에 맞게 추가 메모리가 제공됩니다.

- 가상 머신 인스턴스에 사용할 수 있도록 추가 메모리를 추가합니다.
- 추가 메모리를 추가하여 추가 커널 모듈, 가상 스위치, 모니터링 솔루션 및 기타 추가 백그라운드 작업과 같은 호스트에서 특수 리소스 또는 추가 리소스를 실행합니다.
- NUMA(Non-Uniform Memory Access)를 사용하려는 경우, 256GB의 물리적 RAM이 있는 경우 CPU 소켓 노드당 8GB 또는 소켓 노드당 16GB를 권장합니다.
- 최소 4GB의 스왑 공간을 구성합니다.

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 사용하십시오.

전원 관리

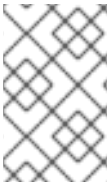
각 컴퓨팅 노드에는 서버 마더보드의 IPMI(Intelligent Platform Management Interface) 기능과 같이 지원되는 전원 관리 인터페이스가 필요합니다.

6.8. CEPH STORAGE 노드 요구 사항

RHOSP(Red Hat OpenStack Platform) director를 사용하여 Red Hat Ceph Storage 노드를 생성하는 경우 추가 요구 사항이 있습니다.

Ceph Storage 노드의 프로세서, 메모리, NIC(네트워크 인터페이스 카드) 및 디스크 레이아웃을 선택하는 방법에 대한 자세한 내용은 *Red Hat Ceph Storage Hardware* 가이드의 Hardware selection recommendations for Red Hat Ceph Storage를 참조하십시오.

각 Ceph Storage 노드에는 서버의 마더보드에 IPMI(Intelligent Platform Management Interface) 기능과 같은 지원되는 전원 관리 인터페이스도 필요합니다.



참고

RHOSP 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.

Ceph Storage 노드 및 RHEL 호환성

- RHOSP 16.2는 RHEL 8.4에서 지원됩니다. 그러나 Ceph Storage 역할 업데이트에 매핑된 호스트는 최신 주요 RHEL 릴리스로 업데이트합니다. RHOSP 16.1 이상으로 업그레이드하기 전에 Red Hat 지식베이스 문서 Red Hat Ceph Storage를 검토하십시오. 지원되는 구성 .

PG(배치 그룹)

- Ceph Storage는 배치 그룹(PG)을 사용하여 대규모의 효율적인 동적 오브젝트 추적을 지원합니다. OSD 오류가 발생하거나 클러스터를 재조정하는 경우 Ceph Storage에서 배치 그룹 및 해당 컨텐츠를 이동하거나 복제할 수 있으므로, Ceph 클러스터의 효율적인 재조정 및 복구가 가능합니다.

- director가 생성하는 기본 배치 그룹 수가 항상 최적의 개수는 아니므로 필요에 따라 올바른 배치 그룹 수를 계산해야 합니다. 배치 그룹 계산기를 사용하여 올바른 개수를 계산할 수 있습니다. PG 계산기를 사용하려면 서비스당 예상 스토리지 사용량을 백분율로 입력하고 개수 OSD와 같은 Ceph 클러스터에 대한 기타 속성을 입력합니다. 계산기는 풀당 최적의 PG 수를 반환합니다. 자세한 내용은 풀 계산기당 PG(배치 그룹)를 참조하십시오.

- 자동 확장은 배치 그룹을 관리하는 대체 방법입니다. 자동 확장 기능을 사용하면 특정 수의 배치 그룹이 아니라 서비스당 예상 Ceph Storage 요구 사항을 백분율로 설정합니다. Ceph는 클러스터 사용 방법에 따라 배치 그룹을 자동으로 확장합니다. 자세한 내용은 *Red Hat Ceph Storage Strategies Guide*의 자동 확장 배치 그룹 을 참조하십시오.

프로세서

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

네트워크 인터페이스 카드

- 최소 1개의 1Gbps NIC(네트워크 인터페이스 카드)가 필요합니다. Red Hat에서는 프로덕션 환경에서 적어도 두 개 이상의 NIC를 사용하도록 권장합니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 NIC를 사용하십시오. 대량의 트래픽에 서비스를 제공하는 RHOSP(Red Hat OpenStack Platform) 환경을 구축하는 경우 특히 스토리지 노드에 10Gbps 인터페이스를 사용합니다.

전원 관리

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

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

6.9. OBJECT STORAGE 노드 요구 사항

Object Storage 노드는 오버클라우드에 오브젝트 스토리지 계층을 제공합니다. Object Storage 프록시는 컨트롤러 노드에 설치됩니다. 스토리지 계층에는 각 노드에 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. 하드 디스크 공간 1TB당 최소 1GB의 메모리를 사용합니다. 최적의 성능을 위해 특히 워크로드가 100GB 미만 파일인 경우 하드 디스크 공간 1TB당 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 저장하려면 SSD의 용량이 더 필요하고, 큰 오브젝트(비디오, 백업)의 경우에는 SSD의 용량을 줄일 수 있습니다.

디스크 레이아웃

권장 노드 구성에는 다음 예제와 비슷한 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 사용하십시오.

전원 관리

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

6.10. 오버클라우드 리포지토리

RHOSP(Red Hat OpenStack Platform) 16.2는 Red Hat Enterprise Linux 8.4에서 실행됩니다. 그러므로 해당 리포지토리의 콘텐츠를 해당하는 Red Hat Enterprise Linux 버전에 고정해야 합니다.

참고

Red Hat Satellite를 사용하여 리포지토리를 동기화하는 경우 특정 버전의 Red Hat Enterprise Linux 리포지토리를 활성화할 수 있습니다. 그러나 리포지토리 레이블은 선택한 버전에도 동일하게 유지됩니다. 예를 들어 8.4 버전의 BaseOS 리포지토리를 활성화하면 리포지토리 이름에 활성화된 특정 버전이 포함되지만 리포지토리 레이블은 여전히 **rhel-8-for-x86_64-baseos-eus-rpms** 입니다.

주의

여기에 지정된 리포지토리만 지원됩니다. 아래 표에 나열되지 않은 제품이나 리포지토리는 별도로 권장되지 않는 한 활성화하지 마십시오. 그러지 않으면 패키지 종속성 문제가 발생할 수 있습니다. EPEL(Extra Packages for Enterprise Linux)을 활성화하지 마십시오.

컨트롤러 노드 리포지토리

다음 표에는 오버클라우드에서 컨트롤러 노드를 위한 코어 리포지토리가 나와 있습니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - BaseOS(RPM) EUS(Extended Update Support)	**rhel-8-for-x86_64-baseos-eus-rpms**	x86_64 시스템용 기본 운영 체제 리포지토리입니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - AppStream(RPM)	**rhel-8-for-x86_64-appstream-eus-rpms**	Red Hat OpenStack Platform 종속성을 포함합니다.
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다.
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64(RPM)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux입니다. 최신 버전의 Ansible을 제공하는 데 사용됩니다.
Red Hat OpenStack Platform 16.2 for RHEL 8(RPM)	**openstack-16.2-for-rhel-8-x86_64-rpms**	코어 Red Hat OpenStack Platform 리포지토리입니다.
Red Hat Fast Datapath for RHEL 8(RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform용 OVS(Open vSwitch) 패키지를 제공합니다.
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPM)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Red Hat Ceph Storage 4 for Red Hat Enterprise Linux 8용 툴입니다.
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-<satellite_version>-for-rhel-8-x86_64-rpms**	Red Hat Satellite 6으로 호스트를 관리하는 툴입니다. 여기서 **<satellite_version>** 은 사용하는 Red Hat Satellite Server 버전입니다.

컴퓨팅 및 ComputeHCI 노드 리포지토리

다음 표에는 오버클라우드의 Compute 및 ComputeHCI 노드를 위한 코어 리포지토리가 나와 있습니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - BaseOS(RPM) EUS(Extended Update Support)	**rhel-8-for-x86_64-baseos-eus-rpms**	x86_64 시스템용 기본 운영 체제 리포지토리입니다.
Red Hat Enterprise Linux 8 for x86_64 - AppStream(RPM)	**rhel-8-for-x86_64-appstream-eus-rpms**	Red Hat OpenStack Platform 종속 패키지를 포함합니다.
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다.

이름	리포지토리	요구 사항 설명
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64(RPM)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux입니다. 최신 버전의 Ansible을 제공하는 데 사용됩니다.
Red Hat OpenStack Platform 16.2 for RHEL 8(RPM)	**openstack-16.2-for-rhel-8-x86_64-rpms**	코어 Red Hat OpenStack Platform 리포지토리입니다.
Red Hat Fast Datapath for RHEL 8(RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform용 OVS(Open vSwitch) 패키지를 제공합니다.
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPM)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Red Hat Ceph Storage 4 for Red Hat Enterprise Linux 8용 툴입니다.
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-<satellite_version>-for-rhel-8-x86_64-rpms**	Red Hat Satellite 6으로 호스트를 관리하는 툴입니다. 여기서 **<satellite_version>** 은 사용하는 Red Hat Satellite Server 버전입니다.

실시간 Compute 리포지토리

다음 표에는 RTC(Real Time Compute) 기능에 사용되는 리포지토리가 나와 있습니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - Real Time(RPM)	**rhel-8-for-x86_64-rt-rpms**	실시간 KVM(RT-KVM) 리포지토리로, 실시간 커널을 활성화하는 패키지가 포함되어 있습니다. RT-KVM을 대상으로 하는 모든 컴퓨팅 노드에 대해 이 리포지토리를 활성화합니다. 알림: 이 리포지토리에 액세스하려면 별도의 **Red Hat OpenStack Platform for Real Time** SKU 서브스크립션이 필요합니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - Real Time for NFV(RPM)	**rhel-8-for-x86_64-nfv-rpms**	NFV용 실시간 KVM(RT-KVM) 리포지토리로, 실시간 커널을 활성화하는 패키지가 포함되어 있습니다. RT-KVM을 대상으로 하는 모든 NFV 노드에 대해 이 리포지토리를 활성화합니다. 알림: 이 리포지토리에 액세스하려면 별도의 **Red Hat OpenStack Platform for Real Time** SKU 서브스크립션이 필요합니다.

Ceph Storage 노드 리포지토리

다음 표에는 오버클라우드의 Ceph Storage 관련 리포지토리가 나열되어 있습니다.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux 8 for x86_64 - BaseOS(RPM)	**rhel-8-for-x86_64-baseos-rpms**	x86_64 시스템용 기본 운영 체제 리포지토리입니다.
Red Hat Enterprise Linux 8 for x86_64 - AppStream(RPM)	**rhel-8-for-x86_64-appstream-rpms**	Red Hat OpenStack Platform 종속 패키지를 포함합니다.
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다. 알림: Ceph Storage 역할에 **overcloud-full** 이미지를 사용한 경우 이 리포지토리를 활성화해야 합니다. Ceph Storage 역할은 이 리포지토리가 필요하지 않은 **overcloud-minimal** 이미지를 사용해야 합니다.
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64(RPM)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux입니다. 최신 버전의 Ansible을 제공하는 데 사용됩니다.
Red Hat OpenStack Platform 16.2 Director Deployment Tools for RHEL 8 x86_64(RPM)	**openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms**	director에서 Ceph Storage 노드를 설정하는 데 사용되는 패키지입니다. 이 리포지토리는 독립 실행형 Ceph Storage 서브스크립션에 포함되어 있습니다. 결합된 OpenStack Platform 및 Ceph Storage 서브스크립션을 사용하는 경우 **openstack-16.2-for-rhel-8-x86_64-rpms** 리포지토리를 사용합니다.

이름	리포지토리	요구 사항 설명
Red Hat OpenStack Platform 16.2 for RHEL 8(RPM)	**openstack-16.2-for-rhel-8-x86_64-rpms**	director에서 Ceph Storage 노드를 설정하는 데 사용되는 패키지입니다. 이 리포지토리는 OpenStack Platform 및 Ceph Storage 서브스크립션에 함께 포함되어 있습니다. 독립 실행형 Ceph Storage 서브스크립션을 사용하는 경우 **openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms** 리포지토리를 사용하십시오.
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPM)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	노드가 Ceph Storage 클러스터와 통신할 수 있는 툴을 제공합니다.
Red Hat Fast Datapath for RHEL 8(RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform용 OVS(Open vSwitch) 패키지를 제공합니다. Ceph Storage 노드에서 OVS를 사용하는 경우 이 리포지토리를 NIC(네트워크 인터페이스 구성) 템플릿에 추가합니다.

IBM POWER 리포지토리

다음 표에는 POWER PC 아키텍처의 RHOSP용 리포지토리가 나와 있습니다. 코어 리포지토리의 리포지토리 대신 이 리포지토리를 사용하십시오.

이름	리포지토리	요구 사항 설명
Red Hat Enterprise Linux for IBM Power, little endian - BaseOS(RPM)	**rhel-8-for-ppc64le-baseos-rpms**	ppc64le 시스템용 기본 운영 체제 리포지토리입니다.
Red Hat Enterprise Linux 8 for IBM Power, little endian - AppStream(RPM)	**rhel-8-for-ppc64le-appstream-rpms**	Red Hat OpenStack Platform 종속 패키지를 포함합니다.
Red Hat Enterprise Linux 8 for IBM Power, little endian - High Availability(RPM)	**rhel-8-for-ppc64le-highavailability-rpms**	Red Hat Enterprise Linux용 고가용성 툴입니다. 컨트롤러 노드 고가용성에 사용됩니다.
Red Hat Fast Datapath for RHEL 8 IBM Power, little endian(RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	OpenStack Platform용 OVS(Open vSwitch) 패키지를 제공합니다.
Red Hat Ansible Engine 2.9 for RHEL 8 IBM Power, little endian(RPM)	**ansible-2.9-for-rhel-8-ppc64le-rpms**	Ansible Engine for Red Hat Enterprise Linux입니다. 최신 버전의 Ansible을 제공하는 데 사용됩니다.

이름	리포지토리	요구 사항 설명
Red Hat OpenStack Platform 16.2 for RHEL 8(RPM)	**openstack-16.2-for-rhel-8-ppc64le-rpms**	ppc64le 시스템용 Red Hat OpenStack Platform 코어 리포지토리입니다.

6.11. 프로비저닝 방법

Red Hat OpenStack Platform 환경의 노드를 프로비저닝하는 데 사용할 수 있는 기본 방법은 다음 세 가지가 있습니다.

Director로 프로비저닝

Red Hat OpenStack Platform director는 표준 프로비저닝 방법입니다. 이 시나리오에서는 **openstack overcloud deploy** 명령을 통해 배포 구성과 프로비저닝 모두를 수행합니다. 표준 프로비저닝 및 배포 방법에 대한 자세한 내용은 7장. *기본 오버클라우드 구성* 을 참조하십시오.

OpenStack Bare Metal(ironic) 서비스로 프로비저닝

이 시나리오에서는 표준 director 배포의 프로비저닝과 구성 단계를 두 개의 개별 프로세스로 나눌 수 있습니다. 표준 director 배포와 관련된 위험을 완화하고 실패 지점을 더 효율적으로 파악하려는 경우 유용합니다. 이 시나리오에 대한 자세한 내용은 8장. *오버클라우드를 배포하기 전에 베어 메탈 노드 프로비저닝*를 참조하십시오.

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 정보 를 참조하십시오.

외부 툴로 프로비저닝

이 시나리오에서는 director가 외부 툴로 사전 프로비저닝한 노드의 오버클라우드 구성을 제어합니다. 전원 관리 제어 없이 오버클라우드를 생성하거나 DHCP/PXE 부트 제한이 있는 네트워크를 사용하려는 경우 또는 QCOW2 **overcloud-full** 이미지에 의존하지 않는 사용자 지정 파티셔닝 레이아웃이 있는 노드를 사용하려는 경우 유용합니다. 이 시나리오에서는 노드 관리에 OpenStack Compute(nova), OpenStack Bare Metal(ironic) 또는 OpenStack Image(glance) 서비스를 사용하지 않습니다.
이 시나리오에 대한 자세한 내용은 9장. *사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 설정*를 참조하십시오.

중요

사전 프로비저닝된 노드와 director 프로비저닝된 노드를 결합할 수 없습니다.

7장. 기본 오버클라우드 구성

기본 설정을 사용하는 오버클라우드에는 사용자 지정 기능이 없습니다. 기본 RHOSP(Red Hat OpenStack Platform) 환경을 구성하려면 다음 작업을 수행해야 합니다.

- 오버클라우드에 사용할 베어 메탈 노드를 등록합니다.
- director에 베어 메탈 노드의 하드웨어 인벤토리를 제공합니다.
- 각 베어 메탈 노드에 지정된 역할에 노드와 일치하는 리소스 클래스를 지정합니다.

작은 정보

이 기본 오버클라우드에 고급 구성 옵션을 추가하고 사양에 맞게 사용자 지정할 수 있습니다. 자세한 내용은 Advanced Overcloud Customization 을 참조하십시오.

7.1. 오버클라우드에 노드 등록

director에는 노드의 하드웨어 및 전원 관리 세부 정보를 지정하는 노드 정의 템플릿이 필요합니다. JSON 형식, **nodes.json** 또는 YAML 형식, **nodes.yaml** 로 이 템플릿을 생성할 수 있습니다.

절차

1. 노드를 나열하는 **nodes.json** 또는 **nodes.yaml** 이라는 템플릿을 생성합니다. 다음 JSON 및 YAML 템플릿 예제를 사용하여 노드 정의 템플릿을 구성하는 방법을 파악합니다.

   JSON 템플릿 예

```
{
  "nodes": [{
    "name": "node01",
    "ports": [{
      "address": "aa:aa:aa:aa:aa:aa",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
    "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"
  },
  {
    "name": "node02",
    "ports": [{
      "address": "bb:bb:bb:bb:bb:bb",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
```

```
        "port_id": "p0"
      }
    }],
    "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"
  }]
}
```

YAML 템플릿 예

```
nodes:
  - name: "node01"
    ports:
      - address: "aa:aa:aa:aa:aa:aa"
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    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"
  - name: "node02"
    ports:
      - address: "bb:bb:bb:bb:bb:bb"
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    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 템플릿에는 다음 속성이 포함되어 있습니다.

name

노드의 논리 이름입니다.

포트

특정 IPMI 장치에 액세스할 수 있는 포트입니다. 다음과 같은 선택적 포트 특성을 정의할 수 있습니다.

- **주소**: 노드에 있는 네트워크 인터페이스의 MAC 주소입니다. 각 시스템의 프로비저닝 NIC에는 MAC 주소만 사용합니다.
- **physical_network**: 프로비저닝 NIC에 연결된 물리적 네트워크입니다.
- **local_link_connection**: IPv6 프로비저닝을 사용하고 LLDP가 인트로스펙션 중에 로컬 링크 연결을 올바르게 채우지 않는 경우 **local_link_connection** 매개변수의 **switch_id** 및 **port_id** 필드에 페이크 데이터를 포함해야 합니다. 페이크 데이터를 포함하는 방법에 대한 자세한 내용은 director 인트로스펙션 사용을 통해 베어 메탈 노드 하드웨어 정보 수집을 참조하십시오.

cpu

(선택 사항) 노드에 있는 CPU 수입니다.

memory

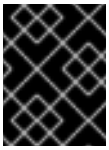
(선택 사항) 메모리 크기(MB)입니다.

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.

pm_type

사용하려는 전원 관리 드라이버. 이 예에서는 IPMI 드라이버(**ipmi**)를 사용합니다.

참고

IPMI는 지원되는 기본 전원 관리 드라이버입니다. 지원되는 전원 관리 유형 및 옵션에 대한 자세한 내용은 전원 관리 드라이버를 참조하십시오. 이러한 전원 관리 드라이버가 예상대로 작동하지 않는 경우 IPMI를 전원 관리에 사용합니다.

pm_user; pm_password

IPMI 사용자 이름 및 암호입니다.

pm_addr

IPMI 장치의 IP 주소입니다.

2. 템플릿을 생성한 후 다음 명령을 실행하여 포멧과 구문을 확인합니다.

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

중요

다중 아키텍처 노드에 **--http-boot /var/lib/ironic/tftpboot/** 옵션도 포함해야 합니다.

3. 파일을 **stack** 사용자의 홈 디렉터리(**/home/stack/nodes.json**)에 저장합니다.

4. 템플릿을 director로 가져와 템플릿의 각 노드를 director로 등록합니다.

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

참고

UEFI 부팅 모드를 사용하는 경우 각 노드에서 부팅 모드도 설정해야 합니다. UEFI 부팅 모드를 설정하지 않고 노드를 세부 검사하면 노드가 기존 모드로 부팅됩니다. 자세한 내용은 부팅 모드 설정을 UEFI 부팅 모드로 설정을 참조하십시오.

5. 노드 등록 및 구성이 완료될 때까지 기다립니다. 완료되면 노드가 director에 성공적으로 등록되어 있는지 확인합니다.

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

7.2. 베어 메탈 노드 하드웨어의 인벤토리 생성

director에는 프로필 태그, 벤치마킹 및 수동 루트 디스크 할당을 위해 RHOSP(Red Hat OpenStack Platform) 배포에 있는 노드의 하드웨어 인벤토리가 필요합니다.

다음 방법 중 하나를 사용하여 하드웨어 인벤토리를 director에 제공할 수 있습니다.

- 자동: 각 노드에서 하드웨어 정보를 수집하는 director의 인트로스펙션 프로세스를 사용할 수 있습니다. 이 프로세스는 각 노드에서 인트로스펙션 에이전트를 시작합니다. 인트로스펙션 에이전트는 노드에서 하드웨어 데이터를 수집하고 그 데이터를 다시 director로 보냅니다. director는 하드웨어 데이터를 언더클라우드 노드의 실행 중인 Object Storage 서비스(swift)에 저장합니다.

- Manual: 각 베어 메탈 머신의 기본 하드웨어 인벤토리를 수동으로 구성할 수 있습니다. 이 인벤토리는 Bare Metal Provisioning 서비스(ironic)에 저장되며 베어 메탈 머신을 관리하고 배포하는 데 사용됩니다.

참고

오버클라우드에 **derive_params.yaml** 을 사용하는 경우 director의 자동 인트로스펙션 프로세스를 사용해야 합니다. 이 경우 인트로스펙션 데이터가 있어야 합니다. **derive_params.yaml** 에 대한 자세한 내용은 워크플로우 및 파생 매개변수를 참조하십시오.

director 자동 인트로스펙션 프로세스는 베어 메탈 프로비저닝 서비스 포트를 설정하는 수동 방법에 비해 다음과 같은 이점을 제공합니다.

- 인트로스펙션은 PXE 부팅에 사용할 포트를 포함하여 하드웨어 정보에 연결된 모든 포트를 기록합니다. (**node.yaml** 에 아직 구성되지 않은 경우).

- 인트로스펙션은 LLDP를 사용하여 속성을 검색할 수 있는 경우 각 포트의 **local_link_connection** 속성을 설정합니다. 수동 방법을 사용하는 경우 노드를 등록할 때 각 포트에 대해 **local_link_connection** 을 구성해야 합니다.
- 인트로스펙션은 스파인-리프형 또는 DCN 아키텍처를 배포할 때 베어 메탈 프로비저닝 서비스 포트에 대한 **physical_network** 속성을 설정합니다.

7.2.1. director 인트로스펙션을 사용하여 베어 메탈 노드 하드웨어 정보 수집

물리적 머신을 베어 메탈 노드로 등록한 후 director 인트로스펙션을 사용하여 하드웨어 세부 정보를 자동으로 추가하고 각 이더넷 MAC 주소에 대한 포트를 생성할 수 있습니다.

작은 정보

자동 인트로스펙션 대신 director에 베어 메탈 노드의 하드웨어 정보를 수동으로 제공할 수 있습니다. 자세한 내용은 베어 메탈 노드 하드웨어 정보 수동 구성을 참조하십시오.

사전 요구 사항

- 오버클라우드에 대한 베어 메탈 노드를 등록합니다.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. pre-introspection 검증 그룹을 실행하여 인트로스펙션 요구 사항을 확인합니다.

```
(undercloud)$ openstack tripleo validator run --group pre-introspection
```

4. 검증 보고서 결과를 확인하십시오.
5. 선택 사항: 특정 검증의 자세한 출력을 확인합니다.

```
(undercloud)$ openstack tripleo validator show run --full <validation>
```

   - & **lt;validation** >를 검토하려는 보고서의 특정 검증 UUID로 바꿉니다.

중요

검증 결과가 **FAILED**이더라도 Red Hat OpenStack Platform 배포나 실행을 방해할 수 없습니다. 그러나 **FAILED** 검증 결과는 프로덕션 환경에서 잠재적으로 문제가 발행할 수 있다는 것을 의미합니다.

6. 각 노드의 하드웨어 속성을 검사합니다. 모든 노드 또는 특정 노드의 하드웨어 속성을 검사할 수 있습니다.
   - 모든 노드의 하드웨어 속성을 검사합니다.

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

- **--all-manageable** 옵션을 사용하여 관리 상태에 있는 노드만 인트로스펙션합니다. 이 예에서는 모든 노드가 관리 상태에 있습니다.
- **--provide** 옵션은 인트로스펙션 이후 모든 노드를 **available** 상태로 리셋합니다.

- 특정 노드의 하드웨어 속성을 검사합니다.

```
(undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
```

- **--provide** 옵션을 사용하여 인트로스펙션 이후 지정된 모든 노드를 **available** 상태로 재설정합니다.
- & **lt;node1** >, **[node 2]** 및 모든 노드를 인트로스펙션하려는 각 노드의 UUID로 바꿉니다.

7. 별도의 터미널 창에서 인트로스펙션 진행 상태 로그를 모니터링합니다.

```
(undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
```

중요

인트로스펙션 프로세스가 완료되었는지 확인합니다. 베어 메탈 노드의 경우 인트로스펙션은 일반적으로 15분 정도 걸립니다. 그러나 세부 검사 네트워크의 크기가 잘못 조정되면 훨씬 더 오래 걸릴 수 있으므로 인트로스펙션이 실패할 수 있습니다.

8. 선택 사항: IPv6를 통해 베어 메탈 프로비저닝에 언더클라우드를 구성한 경우 LLDP가 Bare Metal Provisioning 서비스(ironic) 포트에 대해 **local_link_connection** 을 설정했는지 확인해야 합니다.

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

- 베어 메탈 노드의 포트에 대해 로컬 링크 연결 필드가 비어 있는 경우 페이크 데이터를 수동으로 사용하여 **local_link_connection** 값을 채워야 합니다. 다음 예제에서는 페이크 스위치 ID를 **52:54:00:00:00:00** 으로 설정하고 페이크 포트 ID를 **p0** 으로 설정합니다.

```
(undercloud)$ openstack baremetal port set <port_uuid> \
--local-link-connection switch_id=52:54:00:00:00:00 \
--local-link-connection port_id=p0
```

- 로컬 링크 연결 필드에 페이크 데이터가 포함되어 있는지 확인합니다.

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

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

7.2.2. 베어 메탈 노드 하드웨어 정보 수동 구성

물리적 머신을 베어 메탈 노드로 등록한 후 하드웨어 세부 정보를 수동으로 추가하고 각 이더넷 MAC 주소에 대해 베어 메탈 포트를 생성할 수 있습니다. 오버클라우드를 배포하기 전에 하나 이상의 베어 메탈 포트를 생성해야 합니다.

작은 정보

수동 인트로스펙션 대신 자동 director 인트로스펙션 프로세스를 사용하여 베어 메탈 노드의 하드웨어 정보를 수집할 수 있습니다. 자세한 내용은 director 인트로스펙션을 사용하여 베어 메탈 노드 하드웨어 정보를 수집합니다.

사전 요구 사항

- 오버클라우드에 대한 베어 메탈 노드를 등록합니다.
- node **.json** 에 등록된 노드의 각 포트에 대해 **local_link_connection** 을 구성했습니다. 자세한 내용은 오버클라우드 노드 등록을 참조하십시오.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 노드의 기능에 **boot_option':' local** 을 추가하여 등록된 각 노드의 로컬로 부팅 옵션을 설정합니다.

```
(undercloud)$ openstack baremetal node set \
 --property capabilities="boot_option:local" <node>
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.

4. 배포 커널을 지정하고 노드 드라이버의 램디스크를 배포합니다.

```
(undercloud)$ openstack baremetal node set <node> \
  --driver-info deploy_kernel=<kernel_file> \
  --driver-info deploy_ramdisk=<initramfs_file>
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.
- < **kernel_file** >을 **.kernel** 이미지의 경로로 바꿉니다(예: **file:///var/lib/ironic/httpboot/agent.kernel**).
- < **initramfs_file** >을 **.initramfs** 이미지 경로로 바꿉니다(예: **file:///var/lib/ironic/httpboot/agent.ramdisk**).

5. 노드의 하드웨어 사양과 일치하도록 노드 속성을 업데이트합니다.

```
(undercloud)$ openstack baremetal node set <node> \
  --property cpus=<cpu> \
  --property memory_mb=<ram> \
  --property local_gb=<disk> \
  --property cpu_arch=<arch>
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.
- & **lt;cpu&** gt;를 CPU 수로 바꿉니다.

- &lt **;ram&gt**;을 RAM(MB)으로 바꿉니다.
- &lt **;disk&gt**;를 디스크 크기(GB)로 바꿉니다.
- &lt **;arch&** gt;를 아키텍처 유형으로 바꿉니다.

6. 선택 사항: 각 노드의 IPMI 암호화 제품군을 지정합니다.

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.
- & **lt;version** >을 노드에서 사용할 암호화 제품군 버전으로 바꿉니다. 다음 유효한 값 중 하나로 설정합니다.
  - **3** - 노드는 SHA1 암호화 제품군과 AES-128을 사용합니다.
  - **17** - 노드는 SHA256 암호화 제품군과 함께 AES-128을 사용합니다.

7. 선택 사항: 여러 디스크가 있는 경우 루트 장치 힌트를 설정하여 배포에 사용할 디스크의 배포 램 디스크를 알립니다.

```
(undercloud)$ openstack baremetal node set <node> \
 --property root_device='{"<property>": "<value>"}'
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.
- < **property** > 및 < **value** >를 배포에 사용할 디스크에 대한 세부 정보(예: **root_device='{"size)로 바꿉니다. "128"}'**
  RHOSP에서는 다음 속성을 지원합니다.
  - **모델** (문자열): 장치 식별자.
  - **벤더** (문자열): 장치 벤더.
  - **serial** (문자열): 디스크 일련 번호.
  - **hctl** (문자열): host:Channel:Target: SCSI의 Lun.
  - **크기** (정수): 장치 크기(GB)입니다.
  - **WWN (** 문자열): 고유한 스토리지 식별자.
  - **wwn_with_extension** (문자열): 공급업체 확장이 추가된 고유한 스토리지 식별자입니다.
  - **wwn_vendor_extension** (문자열): 고유한 벤더 스토리지 식별자.
  - **rotational** (부울): 회전 장치(HDD)의 경우 true이며 그렇지 않으면 false(SSD)입니다.
  - **이름** (문자열): 장치 이름(예: /dev/sdb1)은 영구 이름이 있는 장치에만 이 속성을 사용합니다.

참고

둘 이상의 속성을 지정하는 경우 장치가 해당 속성과 일치해야 합니다.If you specify more than one property, the device must match all of those properties.

8. provisioning 네트워크에서 NIC의 MAC 주소로 포트를 생성하여 베어 메탈 프로비저닝 서비스에 노드 네트워크 카드를 알립니다.

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- & **lt;node_uuid&** gt;를 베어 메탈 노드의 고유 ID로 바꿉니다.
- & **lt;mac_address&** gt;를 PXE 부팅에 사용되는 NIC의 MAC 주소로 바꿉니다.

9. 노드의 구성을 검증합니다.

```
(undercloud)$ openstack baremetal node validate <node>
+------------+--------+--------------------------------------------+
| Interface  | Result | Reason                                     |
+------------+--------+--------------------------------------------+
| boot       | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| console    | None   | not supported                              |
| deploy     | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| inspect    | None   | not supported                              |
| management | True   |                                            |
| network    | True   |                                            |
| power      | True   |                                            |
| raid       | True   |                                            |
| storage    | True   |                                            |
+------------+--------+--------------------------------------------+
```

검증 출력 **결과는** 다음과 같습니다.

- **false**: 인터페이스가 검증에 실패했습니다. 제공된 이유로 **instance_info** 매개 변수 **[\'ramdisk', \'kernel' 및 \'image_source']** 가 누락된 경우 Compute 서비스가 배포 프로세스 시작 부분에 누락된 매개변수를 채우므로 이 시점에서 설정되지 않았을 수 있습니다. 전체 디스크 이미지를 사용하는 경우 검증을 전달하도록 **image_source** 만 설정해야 할 수 있습니다.
- **true**: 인터페이스가 검증을 통과했습니다.
- **없음**: 드라이버에서 인터페이스가 지원되지 않습니다.

7.3. 프로필에 노드 태그

각 노드의 하드웨어를 등록 및 검사한 후에 특정 프로필에 노드를 태그합니다. 이러한 프로필 태그에 따라 노드에 일치하는 플레이버(Flavor)를 배포 역할에 지정합니다. 다음 예제는 컨트롤러 노드에 대한 역할, 플레이버, 프로필 및 노드 간의 관계를 보여줍니다.

유형	설명
역할	**Controller** 역할은 director의 컨트롤러 노드 구성 방법을 정의합니다.
플레이버	**control** 플레이버는 노드에서 컨트롤러로 사용할 하드웨어 프로필을 정의합니다. director에서 사용할 노드를 결정할 수 있도록 이 플레이버를 **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 구성합니다.

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

절차

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

```
(undercloud) $ NODE=<NODE NAME OR ID>
(undercloud) $ PROFILE=<PROFILE NAME>
(undercloud) $ openstack baremetal node set --property
capabilities="profile:$PROFILE,boot_option:local" $NODE
```

   - **$NODE** 변수를 노드의 이름 또는 UUID로 설정합니다.
   - **$PROFILE** 변수를 **control** 또는 **compute**와 같은 특정 프로필로 설정합니다.
   - **properties/capabilities**의 **profile** 옵션에는 **profile:control** 또는 **profile:compute** 등 노드에 해당 프로필을 태그 지정하기 위한 **$PROFILE** 변수가 포함되어 있습니다.
   - 각 노드가 부팅되는 방식을 정의하려면 **boot_option:local** 옵션을 설정합니다.

   추가 **openstack baremetal node show** 명령 및 **jq** 필터링을 사용하여 기존 **capabilities** 값을 유지할 수도 있습니다.

```
(undercloud) $ openstack baremetal node set --property
capabilities="profile:$PROFILE,boot_option:local,$(openstack baremetal node show $NODE
-f json -c properties | jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

2. 노드 태그를 완료한 후 할당된 프로필 또는 가능한 프로필을 확인합니다.

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

7.4. 부팅 모드를 UEFI 모드로 설정

기본 부팅 모드는 Legacy BIOS 모드입니다. RHOSP 배포에서 노드를 레거시 BIOS 부팅 모드 대신 UEFI 부팅 모드를 사용하도록 구성할 수 있습니다.

주의

일부 하드웨어는 레거시 BIOS 부팅 모드를 지원하지 않습니다. Legacy BIOS 부팅 모드를 지원하지 않는 하드웨어에서 Legacy BIOS 부팅 모드를 사용하려고 하면 배포에 실패할 수 있습니다. 하드웨어가 성공적으로 배포되었는지 확인하려면 UEFI 부팅 모드를 사용합니다.

참고

UEFI 부팅 모드를 활성화하면 사용자 이미지와 함께 파티션 레이아웃 및 부트로더를 포함하는 자체 전체 디스크 이미지를 빌드해야 합니다. 전체 디스크 이미지 생성에 대한 자세한 내용은 전체 디스크 이미지 생성을 참조하십시오.

절차

1. **undercloud.conf** 파일에 다음 매개변수를 설정합니다.

```
ipxe_enabled = True
```

2. **undercloud.conf** 파일을 저장하고 언더클라우드 설치를 실행합니다.

```
$ openstack undercloud install
```

설치 스크립트가 완료될 때까지 기다리십시오.

3. 등록된 각 노드의 기존 기능을 확인합니다.

```
$ openstack baremetal node show <node> -f json -c properties | jq -r .properties.capabilities
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.

4. 노드의 기존 기능에 **boot_mode: uefi 를 추가하여 각 등록된 노드의 부팅 모드를 uefi로 설정합니다.**

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,<capability_1>,...,
<capability_n>" <node>
```

- & **lt;node&** gt;를 베어 메탈 노드의 ID로 바꿉니다.
- < **capability_1** >, 및 모든 기능을 < **capability_n** >까지 교체합니다.

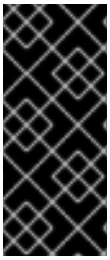
예를 들어 다음 명령을 사용하여 로컬 부팅을 사용하여 부팅 모드를 **uefi로 설정합니다.**

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,boot_option:local"
<node>
```

5. **각 베어 메탈 플레이버의 부팅 모드를 uefi 로 설정합니다.**

```
$ openstack flavor set --property capabilities:boot_mode='uefi' <flavor>
```

7.5. 가상 미디어 부팅 활성화



중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

BMC(Baseboard Management Controller)에서 가상 드라이브 중 하나에 부팅 이미지를 삽입할 수 있도록 **Redfish** 가상 미디어 부팅을 사용하여 노드의 **BMC**에 부팅 이미지를 공급할 수 있습니다. 그 후 노드는 가상 드라이브에서 해당 이미지에 있는 운영 체제로 부팅할 수 있습니다.

Redfish 하드웨어 유형은 가상 미디어를 통한 배포, 복구 및 사용자 이미지 부팅을 지원합니다. **Bare Metal** 서비스(**ironic**)는 노드와 연결된 커널 및 램디스크 이미지를 사용하여 노드 배포 시 **UEFI** 또는 **BIOS** 부팅 모드에 대한 부팅 가능한 **ISO** 이미지를 빌드합니다. 가상 미디어 부팅의 주요 장점은 **PXE**의 **TFTP** 이미지 전송 단계를 제거하고 대신 **HTTP GET** 또는 기타 방법을 사용할 수 있다는 것입니다.

가상 미디어를 통해 **redfish** 하드웨어 유형으로 노드를 부팅하려면 부팅 인터페이스를 **redfish-virtual-media**로 설정하고 **UEFI** 노드의 경우 **ESP(EFI** 시스템 파티션) 이미지를 정의합니다. 다음으로 등록된 노드가 **Redfish** 가상 미디어 부팅을 사용하도록 설정합니다.

사전 요구 사항

- **undercloud.conf** 파일의 **enabled_hardware_types** 매개변수로 활성화된 **Redfish** 드라이버
- 등록된 베어 메탈 노드

- Image 서비스(glance)의 IPA 및 인스턴스 이미지
- UEFI 노드의 경우 Image 서비스(glance)에서 ESP(EFI 시스템 파티션) 이미지 사용 가능
- 베어 메탈 플레이버
- 정리 및 프로비저닝을 위한 네트워크

절차

1. Bare Metal 서비스(ironic) 부팅 인터페이스를 redfish-virtual-media로 설정합니다.

```
$ openstack baremetal node set --boot-interface redfish-virtual-media $NODE_NAME
```

- $ NODE_NAME을 노드 이름으로 바꿉니다.

2. UEFI 노드의 경우 부팅 모드를 uefi로 설정합니다.

```
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
```

- $ NODE를 노드 이름으로 바꿉니다.

참고

BIOS 노드의 경우 이 단계를 완료하지 마십시오.

3. UEFI 노드의 경우 ESP(EFI 시스템 파티션) 이미지를 정의합니다.

```
$ openstack baremetal node set --driver-info bootloader=$ESP $NODE_NAME
```

- $ESP를 glance 이미지 UUID 또는 ESP 이미지 URL로 바꾸고 $NODE_NAME을 노드 이름으로 바꿉니다.

참고

BIOS 노드의 경우 이 단계를 완료하지 마십시오.

4. 베어 메탈 노드에서 포트를 생성하고 베어 메탈 노드에서 NIC의 MAC 주소와 포트를 연결합니다.

```
$ openstack baremetal port create --pxe-enabled True --node $UUID $MAC_ADDRESS
```

- $UUID를 베어 메탈 노드의 UUID로 바꾸고 $MAC_ADDRESS를 베어 메탈 노드에 있는 NIC의 MAC 주소로 바꿉니다.

7.6. 멀티 디스크 클러스터의 ROOT 디스크 정의

대부분의 Ceph Storage 노드는 여러 디스크를 사용합니다. 노드가 여러 디스크를 사용하는 경우 director에서 root 디스크를 식별해야 합니다. 기본적으로 director는 프로비저닝 프로세스 중에 오버클라우드 이미지를 root 디스크에 씁니다.

이 절차를 사용하여 일련 번호로 루트 장치를 식별하십시오. root 디스크를 식별하는 데 사용할 수 있는 기타 속성에 대한 자세한 내용은 7.7절. “root 디스크를 식별하는 속성” 을 참조하십시오.

절차

1. 각 노드의 하드웨어 인트로스펙션에서 디스크 정보를 확인합니다. 노드의 디스크 정보를 표시하는 다음 명령은 다음과 같습니다.

```
(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",
```

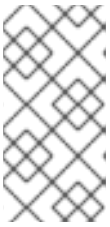
```
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "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. 언더클라우드에서 노드의 **root** 디스크를 설정합니다. **root** 디스크를 정의하는 데 가장 적절한 하드웨어 속성값을 포함시킵니다.

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":"
<serial_number>"}' <node-uuid>
```

예를 들어, **root** 장치를 일련 번호가 **61866da04f380d001ea4e13c12e36ad6**인 **disk 2**로 설정하려면 다음 명령을 입력합니다.

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



참고

선택한 **root** 디스크에서 부팅하도록 각 노드의 **BIOS**를 구성합니다. 먼저 네트워크에서 부팅한 다음 **root** 디스크에서 부팅하도록 부팅 순서를 구성합니다.

director가 **root** 디스크로 사용할 특정 디스크를 식별합니다. **openstack overcloud deploy** 명령을 실

행하면 director가 오버클라우드 이미지를 프로비저닝하고 root 디스크에 씁니다.

7.7. ROOT 디스크를 식별하는 속성

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

- 모델 (문자열): 장치 식별자.
- 벤더 (문자열): 장치 벤더.
- serial (문자열): 디스크 일련 번호.
- hctl (문자열): host:Channel:Target: SCSI의 Lun.
- 크기 (정수): 장치 크기(GB)입니다.
- WWN (문자열): 고유한 스토리지 식별자.
- wwn_with_extension (문자열): 공급업체 확장이 추가된 고유한 스토리지 식별자입니다.
- wwn_vendor_extension (문자열): 고유한 벤더 스토리지 식별자.
- rotational (부울): 회전 장치(HDD)의 경우 true이며 그렇지 않으면 false(SSD)입니다.
- 이름 (문자열): 장치 이름(예: /dev/sdb1)

중요

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

7.8. 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 nodes를 참조하십시오.

참고

RHOSP(Red Hat OpenStack Platform) 서브스크립션에는 OVS(Open vSwitch)가 포함되지만 overcloud-minimal 이미지를 사용하는 경우에는 OVS와 같은 코어 서비스를 사용할 수 없습니다. Ceph Storage 노드를 배포하는 데는 OVS가 필요하지 않습니다. ovs_bond를 사용하여 본드를 정의하는 대신 linux_bond를 사용하십시오. linux_bond에 관한 자세한 내용은 Linux bonding options를 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 Image를 역할의 이름에 추가합니다. 다음 예에서는 Ceph 스토리지 노드의 overcloud-minimal 이미지를 보여줍니다.

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

3. roles_data.yaml 역할 정의 파일에서 rhsm_enforce 매개변수를 False로 설정합니다.

```
rhsm_enforce: False
```

4. openstack overcloud deploy 명령에 환경 파일을 전달합니다.

참고

overcloud-minimal 이미지는 **OVS**가 아닌 표준 **Linux** 브릿지만 지원합니다. **OVS**는 **Red Hat OpenStack Platform** 서브스크립션 인타이틀먼트가 필요한 **OpenStack** 서비스이기 때문입니다.

7.9. 아키텍처별 역할 생성

다중 아키텍처 클라우드를 빌드하는 경우 **roles_data.yaml** 파일에 모든 아키텍처별 역할을 추가해야 합니다. 다음 예제에는 기본 역할과 함께 **ComputePPC64LE** 역할이 포함되어 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
```

Creating a Custom Role File 섹션에는 역할에 관한 정보가 있습니다.

7.10. 환경 파일

언더클라우드에는 오버클라우드 생성 계획을 구성하는 **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. 환경 파일은 필요한 수만큼 추가할 수 있습니다. 그러나 후속 환경 파일에 정의된 매개변수와 리소스가 우선하기 때문에 환경 파일의 순서가 중요합니다. 다음 목록은 환경 파일 순서의 예입니다.

- 각 역할의 노드 수와 플레이버. 오버클라우드 생성에 이 정보를 포함하는 것이 중요합니다.
- 컨테이너화된 **OpenStack** 서비스의 컨테이너 이미지 위치.
- **heat** 템플릿 컬렉션의 초기화 파일(**environments/network-isolation.yaml**)부터 시작하여 네트워크 분리 파일, 사용자 지정 **NIC** 구성 파일, 마지막으로 추가 네트워크 설정 순입니다. 자세한 내용은 ***Advanced Overcloud Customization*** 가이드의 다음 부분을 참조하십시오.
  - **"Basic network isolation"**
  - **"Custom composable networks"**

- "Custom network interface templates"

- 외부 로드 밸런서를 사용하는 경우 모든 외부 로드 밸런싱 환경 파일 자세한 내용은 External Load Balancing for the Overcloud를 참조하십시오.

- Ceph Storage, NFS 또는 iSCSI와 같은 스토리지 환경 파일

- Red Hat CDN 또는 Satellite 등록의 환경 파일

- 기타 사용자 지정 환경 파일

참고

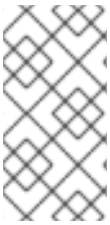
OVN(Open Virtual Networking)은 Red Hat OpenStack Platform 16.2의 기본 네트워킹 메커니즘 드라이버입니다. DVR(분산 가상 라우팅)로 OVN을 사용하려면 openstack overcloud deploy 명령에 environments/services/neutron-ovn-dvr-ha.yaml 파일을 포함해야 합니다. DVR 없이 OVN을 사용하려면 openstack overcloud deploy 명령에 environments/services/neutron-ovn-ha.yaml 파일을 포함해야 합니다.

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

오버클라우드의 고급 기능 사용자 지정에 관한 자세한 내용은 Advanced Overcloud Customization 가이드를 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.



참고

환경 파일 확장자는 .yaml 또는 .template이어야 합니다. 그렇지 않으면 사용자 지정 템플릿 리소스로 처리되지 않습니다.

다음 부분에서는 오버클라우드에 필요한 일부 환경 파일을 생성하는 방법을 설명합니다.

7.11. 노드 수 및 플레이버를 정의하는 환경 생성

기본적으로 **director**는 **baremetal** 플레이버를 사용하여 컨트롤러 노드와 컴퓨팅 노드가 각각 **1**개인 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 컴퓨팅 노드를 각각 **3**개 이상 배포하고 특정 플레이버를 할당하여 노드에 적절한 리소스 사양을 포함합니다. 노드 수와 플레이버 할당을 저장하는 **node-info.yaml**이라는 환경 파일을 생성하려면 다음 단계를 완료합니다.

절차

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

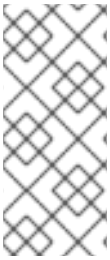
```
(undercloud) $ touch /home/stack/templates/node-info.yaml
```

2. 필요한 노드 수 및 플레이버를 포함하도록 파일을 편집합니다. 이 예제에는 컨트롤러 노드 **3**개와 컴퓨팅 노드 **3**개가 포함되어 있습니다.

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  ControllerCount: 3
  ComputeCount: 3
```

7.12. 언더클라우드 CA를 신뢰하는 환경 파일 생성

언더클라우드가 **TLS**를 사용하고 **CA**(인증 기관)가 공개적으로 신뢰되지 않은 경우, 언더클라우드에서 수행하는 **SSL** 끝점 암호화에 **CA**를 사용할 수 있습니다. 배포의 나머지 부분에서 언더클라우드 엔드포인트에 액세스할 수 있게 하려면 언더클라우드 **CA**를 신뢰하도록 오버클라우드 노드를 구성합니다.



참고

이 방법이 성공하려면 오버클라우드 노드에 언더클라우드의 공용 엔드포인트에 대한 네트워크 경로가 있어야 합니다. 스파인-리프형 네트워킹을 사용하는 배포에는 이 구성을 적용해야 할 수 있습니다.

다음은 언더클라우드에서 사용할 수 있는 두 가지 유형의 사용자 지정 인증서입니다.

- 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 certmonger를 사용하여 자체 로컬 CA를 사용하여 인증서를 생성하는 경우에 적용됩니다. undercloud.conf 파일의 generate_service_certificate 옵션을 사용하여 자동 생성된 인증서를 활성화합니다. 이 예제에서 director는 /etc/pki/ca-trust/source/anchors/cm-local-ca.pem에 CA 인증서를 생성하고, 서버 인증서를 사용하도록 언더클라우드의 HAProxy 인스턴스를 구성합니다. 이 인증서를 OpenStack Platform에 제공하려면 inject-trust-anchor-hiera.yaml 파일에 CA 인증서를 추가합니다.

이 예에서는 /home/stack/ca.crt.pem에 있는 자체 서명 인증서를 사용합니다. 자동 생성 인증서를 사용하는 경우 대신 /etc/pki/ca-trust/source/anchors/cm-local-ca.pem을 사용하십시오.

절차

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.yaml이라는 새 YAML 파일을 만들고, PEM 파일에서 복사한 인증서를 추가합니다.

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

BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
wH
```

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

참고

인증서 문자열이 **PEM** 포멧을 따라야 합니다.

참고

CAMap 매개변수에는 **SSL/TLS** 구성과 관련된 다른 인증서가 포함될 수 있습니다.

오버클라우드를 배포하는 동안 **director**가 **CA** 인증서를 각 오버클라우드 노드에 복사하므로, 각 노드가 언더클라우드의 **SSL** 엔드포인트에서 제공하는 암호화를 신뢰하게 됩니다. 환경 파일에 대한 자세한 내용은 **7.16**절. “오버클라우드 배포에 환경 파일 포함” 의 내용을 참조하십시오.

7.13. 새 배포에서 TSX 비활성화

Red Hat Enterprise Linux 8.3부터 커널은 **Intel TSX(Transactional Synchronization Extensions)** 기능 지원을 기본적으로 비활성화합니다.

워크로드 또는 타사 벤더에 엄격하게 요구되는 경우를 제외하고 새 오버클라우드에 대한 **TSX**를 명시적으로 비활성화해야 합니다.

환경 파일에서 **KernelArgs heat** 매개변수를 설정합니다.

```
parameter_defaults:
  ComputeParameters:
    KernelArgs: "tsx=off"
```

openstack overcloud deploy 명령을 실행하는 경우 이 환경 파일을 포함합니다.

추가 리소스

- **"Guidance on Intel TSX impact on OpenStack guests(RHEL 8.3 이상에 적용)"**

7.14. 배포 명령

OpenStack 환경 생성의 최종 단계는 openstack overcloud deploy 명령을 실행하여 오버클라우드 환경을 생성하는 것입니다. 이 명령을 실행하기 전에 주요 옵션 및 사용자 지정 환경 파일을 포함하는 방법을 숙지하십시오.

주의

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

7.15. 배포 명령 옵션

다음 표에는 openstack overcloud deploy 명령의 추가 매개변수가 나열되어 있습니다.

중요

일부 옵션은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 Red Hat에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에서 사용해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

표 7.1. 배포 명령 옵션

매개변수	설명
--templates [TEMPLATES]	배포하려는 heat 템플릿이 포함된 디렉터리입니다. 비어 있을 경우 배포 명령은 **/usr/share/openstack-tripleo-heat-templates/**를 기본 템플릿 위치로 사용합니다.
--stack STACK	만들거나 업데이트하려는 스택의 이름입니다.
-t [TIMEOUT], --timeout [TIMEOUT]	배포 제한 시간(분)입니다.
--libvirt-type [LIBVIRT_TYPE]	하이퍼바이저에 사용할 가상화 유형입니다.

매개변수	설명
--ntp-server [NTP_SERVER]	시간 동기화에 사용할 NTP(Network Time Protocol) 서버입니다. 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 컨트롤러 노드가 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_proxy**의 사용자 지정 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.
--overcloud-ssh-key OVERCLOUD_SSH_KEY	오버클라우드 노드에 대한 SSH 액세스의 키 경로를 정의합니다.
--overcloud-ssh-network OVERCLOUD_SSH_NETWORK	오버클라우드 노드에 대한 SSH 액세스에 사용할 네트워크 이름을 정의합니다.
-e [EXTRA HEAT TEMPLATE], --environment-file [ENVIRONMENT FILE]	오버클라우드 배포에 전달하려는 추가 환경 파일입니다. 이 옵션을 두 번 이상 지정할 수 있습니다. **openstack overcloud deploy** 명령에 전달하는 환경 파일의 순서가 중요합니다. 예를 들어 순차적으로 전달되는 각 환경 파일의 매개변수는 이전 환경 파일의 동일한 매개변수를 재정의합니다.
--environment-directory	배포에 추가하려는 환경 파일이 들어 있는 디렉터리입니다. 배포 명령은 이러한 환경 파일을 먼저 숫자순으로 처리한 다음, 알파벳순으로 처리합니다.
-r ROLES_FILE	역할 파일을 정의하고 **--templates** 디렉터리의 기본 **roles_data.yaml**을 덮어씁니다. 파일 위치는 절대 경로이거나 **--templates**의 상대 경로일 수 있습니다.
-n NETWORKS_FILE	네트워크 파일을 정의하고 **--templates** 디렉터리의 기본 network_data.yaml을 덮어씁니다. 파일 위치는 절대 경로이거나 **--templates**의 상대 경로일 수 있습니다.
-p PLAN_ENVIRONMENT_FILE	plan 환경 파일을 정의하고 **--templates** 디렉터리의 기본 **plan-environment.yaml**을 덮어씁니다. 파일 위치는 절대 경로이거나 **--templates**의 상대 경로일 수 있습니다.

매개변수	설명
--no-cleanup	배포 후 임시 파일을 삭제하지 않고 해당 위치를 기록하려면 이 옵션을 사용합니다.
--update-plan-only	실제 배포를 수행하지 않고 계획을 업데이트하려면 이 옵션을 사용합니다.
--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. openstack-tripleo-validations
--dry-run	오버클라우드를 생성하지 않고 오버클라우드에서 검증을 수행하려면 이 옵션을 사용합니다.
--run-validations	**openstack-tripleo-validations** 패키지에서 외부 검증을 실행하려면 이 옵션을 사용합니다.
--skip-postconfig	오버클라우드 배포 후 구성을 건너뛰려면 이 옵션을 사용합니다.
--force-postconfig	오버클라우드 배포 후 구성을 강제 적용하려면 이 옵션을 사용합니다.
--skip-deploy-identifier	배포 명령이 **DeployIdentifier** 매개변수의 고유 ID를 생성하지 않게 하려면 이 옵션을 사용합니다. 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.
--disable-password-generation	오버클라우드 서비스의 암호 생성을 비활성화하려면 이 옵션을 사용합니다.
--deployed-server	사전 프로비저닝된 오버클라우드 노드를 배포하려면 이 옵션을 사용합니다. **--disable-validations**와 함께 사용됩니다.

매개변수	설명
--no-config-download, --stack-only	**config-download** 워크플로우를 비활성화하고 스택과 관련 OpenStack 리소스만 생성하려면 이 옵션을 사용합니다. 이 명령은 오버클라우드에 소프트웨어 구성을 적용하지 않습니다.
--config-download-only	오버클라우드 스택 생성을 비활성화하고 **config-download** 워크플로우만 실행하여 소프트웨어 구성을 적용하려면 이 옵션을 사용합니다.
--output-dir OUTPUT_DIR	저장된 **config-download** 출력에 사용할 디렉터리입니다. 이 디렉터리는 mistral 사용자만 쓸 수 있어야 합니다. 지정되지 않으면 director에서 기본값인 **/var/lib/mistral/overcloud**를 사용합니다.
--override-ansible-cfg OVERRIDE_ANSIBLE_CFG	Ansible 설정 파일의 경로입니다. 파일의 설정이 **config-download**에서 기본적으로 생성하는 설정을 덮어씁니다.
--config-download-timeout CONFIG_DOWNLOAD_TIMEOUT	**config-download** 단계에 사용하려는 제한 시간(분)입니다. 설정되지 않으면 director는 스택 배포 작업 후에 **--timeout** 매개변수에서 남은 시간을 기본값으로 설정합니다.
--limit NODE1,NODE2	config-download 플레이북 실행을 특정 노드 또는 노드 세트로 제한하려면 쉼표로 구분된 노드 목록과 함께 이 옵션을 사용합니다. 예를 들어 **--limit** 옵션은 확장 작업 중 새 노드에서만 config-download를 실행하려는 경우 유용합니다. 이 인수로 인해 호스트 간 인스턴스 실시간 마이그레이션이 실패할 수 있습니다. ansible-playbook-command.sh 스크립트를 사용하여 config-download 실행을 참조하십시오.
--tags TAG1,TAG2	**(기술 프리뷰)** 이 옵션을 config-download 플레이북의 콤마로 구분된 태그 목록과 함께 사용하여 특정 config-download 작업 세트를 통해 배포를 실행합니다.
--skip-tags TAG1,TAG2	**(기술 프리뷰)** 이 옵션을 config-download 플레이북에서 건너뛰고자 하는 태그 목록 (콤마로 구분)과 함께 사용합니다.

전체 옵션 목록을 보려면 다음 명령을 실행합니다.

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

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 7.2. 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 Block Storage(cinder) 노드 수	**BlockStorageCount**
--swift-storage-scale	확장할 Object Storage(swift) 노드 수	**ObjectStorageCount**
--control-flavor	컨트롤러 노드에 사용하려는 플레이버	**OvercloudControllerFlavor**
--compute-flavor	컴퓨팅 노드에 사용하려는 플레이버	**OvercloudComputeFlavor**
--ceph-storage-flavor	Ceph Storage 노드에 사용하려는 플레이버	**OvercloudCephStorageFlavor**
--block-storage-flavor	Block Storage(cinder) 노드에 사용하려는 플레이버	**OvercloudBlockStorageFlavor**
--swift-storage-flavor	Object Storage(swift) 노드에 사용하려는 플레이버	**OvercloudSwiftStorageFlavor**
--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
--disable-validations	사전 배포 검증을 완전히 비활성화합니다. 이러한 검증은 **openstack-tripleo-validations** 패키지에서 외부 검증으로 대체된 기본 제공 사전 배포 검증입니다.	매개변수 매핑 없음

매개변수	설명	heat 템플릿 매개변수
--config-download	**config-download** 메커니즘을 사용하여 배포를 실행합니다. 이제 이 값이 기본값이며 이 CLI 옵션은 나중에 제거될 수 있습니다.	매개변수 매핑 없음
--rhel-reg	오버클라우드 노드를 고객 포털 또는 Satellite 6에 등록하려면 이 옵션을 사용합니다.	**RhsmVars**
--reg-method	오버클라우드 노드에 사용할 등록 방법을 정의하려면 이 옵션을 사용합니다. Red Hat Satellite 6 또는 Red Hat Satellite 5에는 **satellite**를 고객 포털에는 **portal**을 사용합니다.	**RhsmVars**
--reg-org [REG_ORG]	등록에 사용하려는 조직입니다.	**RhsmVars**
--reg-force	시스템이 이미 등록되어 있어도 시스템을 등록하려면 이 옵션을 사용합니다.	**RhsmVars**
--reg-sat-url [REG_SAT_URL]	오버클라우드 노드를 등록할 Satellite 서버의 기본 URL입니다. Satellite 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 다음, **katello-agent**를 설치합니다. Red Hat Satellite 5 서버인 경우 오버클라우드는 **RHN-ORG-TRUSTED-SSL-CERT** 파일을 가져오고 **rhnreg_ks**에 등록합니다.	**RhsmVars**
--reg-activation-key [REG_ACTIVATION_KEY]	등록에 사용할 활성화 키를 정의하려면 이 옵션을 사용합니다.	**RhsmVars**

이러한 매개변수는 **Red Hat OpenStack Platform**의 이후 버전에서 삭제될 예정입니다.

7.16. 오버클라우드 배포에 환경 파일 포함

-e 옵션으로 환경 파일을 포함하여 오버클라우드를 사용자 지정할 수 있습니다. 환경 파일은 필요한 수만큼 추가할 수 있습니다. 그러나 후속 환경 파일에 정의된 매개변수와 리소스가 우선하기 때문에 환경 파일의 순서가 중요합니다.

-e 옵션을 사용하여 오버클라우드에 추가하는 환경 파일은 오버클라우드 스택 정의의 일부가 됩니다.

다음 명령은 이 시나리오의 앞부분에서 정의한 환경 파일을 사용하여 오버클라우드 생성을 시작하는 방법의 예제입니다.

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/node-info.yaml\
  -e /home/stack/containers-prepare-parameter.yaml \
  -e /home/stack/inject-trust-anchor-hiera.yaml \
  -r /home/stack/templates/roles_data.yaml \
```

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.

-e /home/stack/containers-prepare-parameter.yaml

컨테이너 이미지 준비 환경 파일을 추가합니다. 언더클라우드 설치 중에 이 파일이 생성되었으며, 오버클라우드 생성 시 동일한 파일을 사용할 수 있습니다.

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

환경 파일을 추가하여 언더클라우드에 사용자 지정 인증서를 설치합니다.

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

(선택 사항) 사용자 지정 역할을 사용하거나 다중 아키텍처 클라우드를 사용하려는 경우 생성된

역할 데이터입니다. 자세한 내용은 7.9절. “아키텍처별 역할 생성” 의 내용을 참조하십시오.

director를 사용하려면 재배포 및 배포 후 기능을 위해 이러한 환경 파일이 필요합니다. 이러한 파일을 포함하지 않으면 오버클라우드가 손상될 수 있습니다.

이후 단계에서 오버클라우드 구성을 수정하려면 다음 작업을 수행합니다.

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

오버클라우드 스택을 업데이트할 때 director가 수동 설정을 덮어쓰므로 오버클라우드 설정을 직접 편집하지 마십시오.

7.17. 사전 배포 검증 실행

pre-deployment 검증 그룹을 실행하여 배포 요구 사항을 확인합니다.

절차

1. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 이 검증을 수행하려면 오버클라우드 계획의 사본이 필요합니다. 필요한 모든 환경 파일과 함께 오버클라우드 계획을 업로드합니다. 계획만 업로드하려면 --update-plan-only 옵션을 사용하여 openstack overcloud deploy 명령을 실행하십시오.

```
$ openstack overcloud deploy --templates \
    -e environment-file1.yaml \
    -e environment-file2.yaml \
    ...
    --update-plan-only
```

3. --group pre-deployment 옵션을 사용하여 openstack tripleo validator run 명령을 실행합니다.

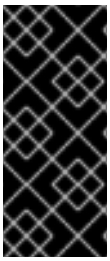
```
$ openstack tripleo validator run --group pre-deployment
```

4. 오버클라우드에서 기본 계획 이름 **overcloud** 이외의 다른 계획 이름을 사용하려면 다음과 같이 **--plan** 옵션을 사용하여 계획 이름을 설정합니다.

```
$ openstack tripleo validator run --group pre-deployment \
  --plan myovercloud
```

5. 검증 보고서 결과를 확인하십시오. 특정 검증의 자세한 출력을 보려면 보고서의 특정 검증 **UUID**에 대해 **openstack tripleo validator show run --full** 명령을 실행합니다.

```
$ openstack tripleo validator show run --full <UUID>
```



중요

검증 결과가 **FAILED**이더라도 **Red Hat OpenStack Platform** 배포나 실행을 방해할 수 없습니다. 그러나 **FAILED** 검증 결과는 프로덕션 환경에서 잠재적으로 문제가 발행할 수 있다는 것을 의미합니다.

7.18. 오버클라우드 배포 출력

오버클라우드 생성이 완료되면 **director**가 오버클라우드를 설정하기 위해 실행한 **Ansible** 플레이 개요를 표시합니다.

```
PLAY RECAP *********************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
===============================================================================
```

director는 또한 오버클라우드 액세스에 필요한 세부 정보도 제공합니다.

```
Ansible passed.
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
```

7.19. 오버클라우드 액세스

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

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

이 명령은 언더클라우드 **CLI**에서 오버클라우드와 상호 작용하는 데 필요한 환경 변수를 로드합니다. 명령 프롬프트가 다음과 같이 변경됩니다.

```
(overcloud) $
```

언더클라우드와 상호 작용으로 돌아가려면 다음 명령을 실행합니다.

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

7.20. 배포 후 검증 실행

post-deployment 검증 그룹을 실행하여 배포 후 상태를 확인합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. **--group post-deployment** 옵션을 사용하여 **openstack tripleo validator run** 명령을 실행합니다.

```
$ openstack tripleo validator run --group post-deployment
```

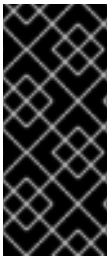
3. 오버클라우드에서 기본 계획 이름 **overcloud** 이외의 다른 계획 이름을 사용하려면 다음과 같이 **--plan** 옵션을 사용하여 계획 이름을 설정합니다.

```
$ openstack tripleo validator run --group post-deployment \
  --plan myovercloud
```

4.

검증 보고서 결과를 확인하십시오. 특정 검증의 자세한 출력을 보려면 보고서의 특정 검증 **UUID**에 대해 **openstack tripleo validator show run --full** 명령을 실행합니다.

```
$ openstack tripleo validator show run --full <UUID>
```



중요

검증 결과가 **FAILED**이더라도 **Red Hat OpenStack Platform** 배포나 실행을 방해할 수 없습니다. 그러나 **FAILED** 검증 결과는 프로덕션 환경에서 잠재적으로 문제가 발행할 수 있다는 것을 의미합니다.

8장. 오버클라우드를 배포하기 전에 베어 메탈 노드 프로비저닝

중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

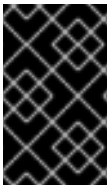
오버클라우드 배포 프로세스에는 다음과 같은 두 개의 기본 작업이 포함되어 있습니다.

- 노드 프로비저닝
- 오버클라우드 배포

이 작업을 개별 프로세스로 나누면 이 프로세스와 관련된 위험을 완화하고 실패 지점을 더 효율적으로 파악할 수 있습니다.

1. 베어 메탈 노드를 프로비저닝합니다.
   a. **yaml** 형식으로 노드 정의 파일을 작성합니다.
   b. 노드 정의 파일을 포함하여 프로비저닝 명령을 실행합니다.
2. 오버클라우드를 배포합니다.
   a. 프로비저닝 명령에서 생성하는 **Heat** 환경 파일을 포함하여 배포 명령을 실행합니다.

프로비저닝 프로세스는 노드를 프로비저닝하고, 노드 개수, 예측 노드 배치, 사용자 지정 이미지, 사용자 지정 **NIC**를 비롯하여 다양한 노드 사양을 포함하는 **Heat** 환경 파일을 생성합니다. 오버클라우드를 배포할 때 배포 명령에 이 파일을 추가하십시오.



중요

사전 프로비저닝된 노드와 **director** 프로비저닝된 노드를 결합할 수 없습니다.

8.1. 오버클라우드에 노드 등록

director에는 노드의 하드웨어 및 전원 관리 세부 정보를 지정하는 노드 정의 템플릿이 필요합니다. **JSON** 형식, **nodes.json** 또는 **YAML** 형식, **nodes.yaml** 로 이 템플릿을 생성할 수 있습니다.

절차

1. 노드를 나열하는 **nodes.json** 또는 **nodes.yaml** 이라는 템플릿을 생성합니다. 다음 **JSON** 및 **YAML** 템플릿 예제를 사용하여 노드 정의 템플릿을 구성하는 방법을 파악합니다.

JSON 템플릿 예

```
{
  "nodes": [{
    "name": "node01",
    "ports": [{
      "address": "aa:aa:aa:aa:aa:aa",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
    "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"
  },
  {
    "name": "node02",
    "ports": [{
      "address": "bb:bb:bb:bb:bb:bb",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
```

```
    "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"
  }]
}
```

YAML 템플릿 예

```
nodes:
  - name: "node01"
    ports:
      - address: "aa:aa:aa:aa:aa:aa"
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    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"
  - name: "node02"
    ports:
      - address: "bb:bb:bb:bb:bb:bb"
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    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 템플릿에는 다음 속성이 포함되어 있습니다.

name

노드의 논리 이름입니다.

포트

특정 IPMI 장치에 액세스할 수 있는 포트입니다. 다음과 같은 선택적 포트 특성을 정의할 수 있습니다.

- 주소: 노드에 있는 네트워크 인터페이스의 MAC 주소입니다. 각 시스템의 프로비저닝 NIC에는 MAC 주소만 사용합니다.
- physical_network: 프로비저닝 NIC에 연결된 물리적 네트워크입니다.
- local_link_connection: IPv6 프로비저닝을 사용하고 LLDP가 인트로스펙션 중에 로컬 링크 연결을 올바르게 채우지 않는 경우 local_link_connection 매개변수의 switch_id 및 port_id 필드에 페이크 데이터를 포함해야 합니다. 페이크 데이터를 포함하는 방법에 대한 자세한 내용은 director 인트로스펙션 사용을 통해 베어 메탈 노드 하드웨어 정보 수집을 참조하십시오.

cpu

(선택 사항) 노드에 있는 CPU 수입니다.

memory

(선택 사항) 메모리 크기(MB)입니다.

disk

(선택 사항) 하드 디스크의 크기(GB)입니다.

arch

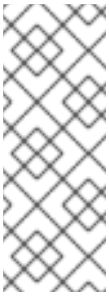
(선택 사항) 시스템 아키텍처입니다.

중요

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

pm_type

사용하려는 전원 관리 드라이버. 이 예에서는 **IPMI** 드라이버(**ipmi**)를 사용합니다.



참고

IPMI는 지원되는 기본 전원 관리 드라이버입니다. 지원되는 전원 관리 유형 및 옵션에 대한 자세한 내용은 전원 관리 드라이버를 참조하십시오. 이러한 전원 관리 드라이버가 예상대로 작동하지 않는 경우 **IPMI**를 전원 관리에 사용합니다.

pm_user; pm_password

IPMI 사용자 이름 및 암호입니다.

pm_addr

IPMI 장치의 **IP** 주소입니다.

2. 템플릿을 생성한 후 다음 명령을 실행하여 포멧과 구문을 확인합니다.

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

중요

다중 아키텍처 노드에 **--http-boot /var/lib/ironic/tftpboot/** 옵션도 포함해야 합니다.

3. 파일을 **stack** 사용자의 홈 디렉터리(**/home/stack/nodes.json**)에 저장합니다.

4. 템플릿을 **director**로 가져와 템플릿의 각 노드를 **director**로 등록합니다.

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

참고

UEFI 부팅 모드를 사용하는 경우 각 노드에서 부팅 모드도 설정해야 합니다. **UEFI** 부팅 모드를 설정하지 않고 노드를 세부 검사하면 노드가 기존 모드로 부팅됩니다. 자세한 내용은 부팅 모드 설정을 **UEFI** 부팅 모드로 설정을 참조하십시오.

5. 노드 등록 및 구성이 완료될 때까지 기다립니다. 완료되면 노드가 **director**에 성공적으로 등록되어 있는지 확인합니다.

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

8.2. 베어 메탈 노드 하드웨어의 인벤토리 생성

director에는 프로필 태그, 벤치마킹 및 수동 루트 디스크 할당을 위해 **RHOSP(Red Hat OpenStack Platform)** 배포에 있는 노드의 하드웨어 인벤토리가 필요합니다.

다음 방법 중 하나를 사용하여 하드웨어 인벤토리를 **director**에 제공할 수 있습니다.

- 자동: 각 노드에서 하드웨어 정보를 수집하는 **director**의 인트로스펙션 프로세스를 사용할 수 있습니다. 이 프로세스는 각 노드에서 인트로스펙션 에이전트를 시작합니다. 인트로스펙션 에이전트는 노드에서 하드웨어 데이터를 수집하고 그 데이터를 다시 **director**로 보냅니다. **director**는 하드웨어 데이터를 언더클라우드 노드의 실행 중인 **Object Storage** 서비스(**swift**)에 저장합니다.

- **Manual:** 각 베어 메탈 머신의 기본 하드웨어 인벤토리를 수동으로 구성할 수 있습니다. 이 인벤토리는 **Bare Metal Provisioning** 서비스(**ironic**)에 저장되며 베어 메탈 머신을 관리하고 배포하는 데 사용됩니다.

참고

오버클라우드에 **derive_params.yaml** 을 사용하는 경우 **director**의 자동 인트로스펙션 프로세스를 사용해야 합니다. 이 경우 인트로스펙션 데이터가 있어야 합니다. **derive_params.yaml** 에 대한 자세한 내용은 워크플로우 및 파생 매개변수를 참조하십시오.

director 자동 인트로스펙션 프로세스는 베어 메탈 프로비저닝 서비스 포트를 설정하는 수동 방법에 비해 다음과 같은 이점을 제공합니다.

- 인트로스펙션은 **PXE** 부팅에 사용할 포트를 포함하여 하드웨어 정보에 연결된 모든 포트를 기록합니다. (**node.yaml** 에 아직 구성되지 않은 경우).

- 인트로스펙션은 **LLDP**를 사용하여 속성을 검색할 수 있는 경우 각 포트의 **local_link_connection** 속성을 설정합니다. 수동 방법을 사용하는 경우 노드를 등록할 때 각 포트에 대해 **local_link_connection** 을 구성해야 합니다.

- 인트로스펙션은 스파인-리프형 또는 **DCN** 아키텍처를 배포할 때 베어 메탈 프로비저닝 서비스 포트에 대한 **physical_network** 속성을 설정합니다.

8.2.1. director 인트로스펙션을 사용하여 베어 메탈 노드 하드웨어 정보 수집

물리적 머신을 베어 메탈 노드로 등록한 후 **director** 인트로스펙션을 사용하여 하드웨어 세부 정보를 자동으로 추가하고 각 이더넷 **MAC** 주소에 대한 포트를 생성할 수 있습니다.

작은 정보

자동 인트로스펙션 대신 **director**에 베어 메탈 노드의 하드웨어 정보를 수동으로 제공할 수 있습니다. 자세한 내용은 베어 메탈 노드 하드웨어 정보 수동 구성을 참조하십시오.

사전 요구 사항

- 오버클라우드에 대한 베어 메탈 노드를 등록합니다.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **pre-introspection** 검증 그룹을 실행하여 인트로스펙션 요구 사항을 확인합니다.

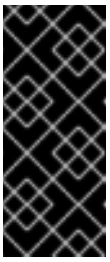
```
(undercloud)$ openstack tripleo validator run --group pre-introspection
```

4. 검증 보고서 결과를 확인하십시오.

5. 선택 사항: 특정 검증의 자세한 출력을 확인합니다.

```
(undercloud)$ openstack tripleo validator show run --full <validation>
```

- & lt;validation >를 검토하려는 보고서의 특정 검증 UUID로 바꿉니다.



중요

검증 결과가 FAILED이더라도 Red Hat OpenStack Platform 배포나 실행을 방해할 수 없습니다. 그러나 FAILED 검증 결과는 프로덕션 환경에서 잠재적으로 문제가 발행할 수 있다는 것을 의미합니다.

6. 각 노드의 하드웨어 속성을 검사합니다. 모든 노드 또는 특정 노드의 하드웨어 속성을 검사할 수 있습니다.

- 모든 노드의 하드웨어 속성을 검사합니다.

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

  - --all-manageable 옵션을 사용하여 관리 상태에 있는 노드만 인트로스펙션합니다. 이 예에서는 모든 노드가 관리 상태에 있습니다.

  - --provide 옵션은 인트로스펙션 이후 모든 노드를 available 상태로 리셋합니다.

- 특정 노드의 하드웨어 속성을 검사합니다.

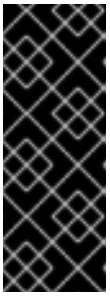
```
(undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
```

  - --provide 옵션을 사용하여 인트로스펙션 이후 지정된 모든 노드를 available 상태로 재설정합니다.

- & lt;node1 > , [node 2] 및 모든 노드를 인트로스펙션하려는 각 노드의 UUID로 바꿉니다.

7. 별도의 터미널 창에서 인트로스펙션 진행 상태 로그를 모니터링합니다.

```
(undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
```



중요

인트로스펙션 프로세스가 완료되었는지 확인합니다. 베어 메탈 노드의 경우 인트로스펙션은 일반적으로 15분 정도 걸립니다. 그러나 세부 검사 네트워크의 크기가 잘못 조정되면 훨씬 더 오래 걸릴 수 있으므로 인트로스펙션이 실패할 수 있습니다.

8. 선택 사항: IPv6를 통해 베어 메탈 프로비저닝에 언더클라우드를 구성한 경우 LLDP가 Bare Metal Provisioning 서비스(ironic) 포트에 대해 local_link_connection 을 설정했는지 확인해야 합니다.

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link
Connection"
```

- 베어 메탈 노드의 포트에 대해 로컬 링크 연결 필드가 비어 있는 경우 페이크 데이터를 수동으로 사용하여 local_link_connection 값을 채워야 합니다. 다음 예제에서는 페이크 스위치 ID를 52:54:00:00:00:00 으로 설정하고 페이크 포트 ID를 p0 으로 설정합니다.

```
(undercloud)$ openstack baremetal port set <port_uuid> \
--local-link-connection switch_id=52:54:00:00:00:00 \
--local-link-connection port_id=p0
```

- 로컬 링크 연결 필드에 페이크 데이터가 포함되어 있는지 확인합니다.

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link
Connection"
```

인트로스펙션이 완료되면 모든 노드가 available 상태로 변경됩니다.

8.2.2. 베어 메탈 노드 하드웨어 정보 수동 구성

물리적 머신을 베어 메탈 노드로 등록한 후 하드웨어 세부 정보를 수동으로 추가하고 각 이더넷 **MAC** 주소에 대해 베어 메탈 포트를 생성할 수 있습니다. 오버클라우드를 배포하기 전에 하나 이상의 베어 메탈 포트를 생성해야 합니다.

작은 정보

수동 인트로스펙션 대신 자동 **director** 인트로스펙션 프로세스를 사용하여 베어 메탈 노드의 하드웨어 정보를 수집할 수 있습니다. 자세한 내용은 **director** 인트로스펙션을 사용하여 베어 메탈 노드 하드웨어 정보를 수집합니다.

사전 요구 사항

- 오버클라우드에 대한 베어 메탈 노드를 등록합니다.
- **node .json** 에 등록된 노드의 각 포트에 대해 **local_link_connection** 을 구성했습니다. 자세한 내용은 오버클라우드 노드 등록을 참조하십시오.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 노드의 기능에 **boot_option':' local** 을 추가하여 등록된 각 노드의 로컬로 부팅 옵션을 설정합니다.

```
(undercloud)$ openstack baremetal node set \
 --property capabilities="boot_option:local" <node>
```

- **& lt;node& gt;**를 베어 메탈 노드의 **ID**로 바꿉니다.

4. 배포 커널을 지정하고 노드 드라이버의 램디스크를 배포합니다.

```
(undercloud)$ openstack baremetal node set <node> \
  --driver-info deploy_kernel=<kernel_file> \
  --driver-info deploy_ramdisk=<initramfs_file>
```

- & lt;node& gt;를 베어 메탈 노드의 ID로 바꿉니다.
- < kernel_file >을 .kernel 이미지의 경로로 바꿉니다(예: file:///var/lib/ironic/httpboot/agent.kernel).
- < initramfs_file >을 .initramfs 이미지 경로로 바꿉니다(예: file:///var/lib/ironic/httpboot/agent.ramdisk).

5. 노드의 하드웨어 사양과 일치하도록 노드 속성을 업데이트합니다.

```
(undercloud)$ openstack baremetal node set <node> \
 --property cpus=<cpu> \
 --property memory_mb=<ram> \
 --property local_gb=<disk> \
 --property cpu_arch=<arch>
```

- & lt;node& gt;를 베어 메탈 노드의 ID로 바꿉니다.
- & lt;cpu& gt;를 CPU 수로 바꿉니다.
- &lt ;ram>을 RAM(MB)으로 바꿉니다.
- &lt ;disk>를 디스크 크기(GB)로 바꿉니다.
- &lt ;arch& gt;를 아키텍처 유형으로 바꿉니다.

6. 선택 사항: 각 노드의 IPMI 암호화 제품군을 지정합니다.

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

- & lt;node& gt;를 베어 메탈 노드의 ID로 바꿉니다.

- & lt;version >을 노드에서 사용할 암호화 제품군 버전으로 바꿉니다. 다음 유효한 값 중 하나로 설정합니다.
  - 3 - 노드는 SHA1 암호화 제품군과 AES-128을 사용합니다.
  - 17 - 노드는 SHA256 암호화 제품군과 함께 AES-128을 사용합니다.

7. 선택 사항: 여러 디스크가 있는 경우 루트 장치 힌트를 설정하여 배포에 사용할 디스크의 배포 램디스크를 알립니다.

```
(undercloud)$ openstack baremetal node set <node> \
 --property root_device='{"<property>": "<value>"}'
```

- & lt;node& gt;를 베어 메탈 노드의 ID로 바꿉니다.
- < property > 및 < value >를 배포에 사용할 디스크에 대한 세부 정보(예: root_device='{"size)로 바꿉니다. "128"}'

  RHOSP에서는 다음 속성을 지원합니다.
  - 모델 (문자열): 장치 식별자.
  - 벤더 (문자열): 장치 벤더.
  - serial (문자열): 디스크 일련 번호.
  - hctl (문자열): host:Channel:Target: SCSI의 Lun.
  - 크기 (정수): 장치 크기(GB)입니다.
  - WWN (문자열): 고유한 스토리지 식별자.

- **wwn_with_extension (문자열): 공급업체 확장이 추가된 고유한 스토리지 식별자입니다.**
- **wwn_vendor_extension (문자열): 고유한 벤더 스토리지 식별자.**
- **rotational (부울): 회전 장치(HDD)의 경우 true이며 그렇지 않으면 false(SSD)입니다.**
- **이름 (문자열): 장치 이름(예: /dev/sdb1)은 영구 이름이 있는 장치에만 이 속성을 사용합니다.**

참고

둘 이상의 속성을 지정하는 경우 장치가 해당 속성과 일치해야 합니다.**If you specify more than one property, the device must match all of those properties.**

8. **provisioning 네트워크에서 NIC의 MAC 주소로 포트를 생성하여 베어 메탈 프로비저닝 서비스에 노드 네트워크 카드를 알립니다.**

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- **& lt;node_uuid& gt;를 베어 메탈 노드의 고유 ID로 바꿉니다.**
- **& lt;mac_address& gt;를 PXE 부팅에 사용되는 NIC의 MAC 주소로 바꿉니다.**

9. **노드의 구성을 검증합니다.**

```
(undercloud)$ openstack baremetal node validate <node>
+------------+--------+--------------------------------------------+
| Interface  | Result | Reason                                     |
+------------+--------+--------------------------------------------+
| boot       | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
```

```
| console    | None   | not supported                                |
| deploy     | False  | Cannot validate image information for node   |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b         |
|            |        | because one or more parameters are missing   |
|            |        | from its instance_info. Missing are:         |
|            |        | ['ramdisk', 'kernel', 'image_source']        |
| inspect    | None   | not supported                                |
| management | True   |                                              |
| network    | True   |                                              |
| power      | True   |                                              |
| raid       | True   |                                              |
| storage    | True   |                                              |
+------------+--------+----------------------------------------------+
```

검증 출력 결과는 다음과 같습니다.

- **false:** 인터페이스가 검증에 실패했습니다. 제공된 이유로 **instance_info** 매개 변수 **[\'ramdisk', \'kernel'** 및 **\'image_source']** 가 누락된 경우 **Compute** 서비스가 배포 프로세스 시작 부분에 누락된 매개변수를 채우므로 이 시점에서 설정되지 않았을 수 있습니다. 전체 디스크 이미지를 사용하는 경우 검증을 전달하도록 **image_source** 만 설정해야 할 수 있습니다.
- **true:** 인터페이스가 검증을 통과했습니다.
- 없음: 드라이버에서 인터페이스가 지원되지 않습니다.

8.3. 베어 메탈 노드 프로비저닝

새 **YAML** 파일 **~/overcloud-baremetal-deploy.yaml**을 생성하고, 배포할 베어 메탈 노드의 수량과 속성을 정의한 다음 오버클라우드 역할을 이 노드에 할당합니다. 프로비저닝 프로세스에서 **openstack overcloud deploy** 명령에 포함할 수 있는 **heat** 환경 파일을 생성합니다.

사전 요구 사항

- 언더클라우드가 설치되어 있어야 합니다. 자세한 내용은 **director** 설치를 참조하십시오.
- 베어 메탈 노드는 세부 검사되어 프로비저닝 및 배포에 사용할 수 있습니다. 자세한 내용은 오버클라우드 노드 등록 및 베어 메탈 노드 하드웨어 인벤토리 생성을 참조하십시오.

절차

1. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 새 ~/**overcloud-baremetal-deploy.yaml** 파일을 생성하고 프로비저닝할 각 역할의 노드 개수를 정의합니다. 예를 들어 세 개의 컨트롤러 노드와 세 개의 컴퓨팅 노드를 프로비저닝하려면 다음 구문을 사용하십시오.

```
- name: Controller
  count: 3
- name: Compute
  count: 3
```

3. ~/**overcloud-baremetal-deploy.yaml** 파일에서 노드에 할당할 예측 가능한 노드 배치, 사용자 지정 이미지, 사용자 지정 **NIC** 또는 기타 속성을 정의합니다. 예를 들어, 다음 예제 구문을 사용하여 **node00, node01** 및 **node02** 노드에 세 개의 컨트롤러 노드와 **node04, node05** 및 **node06**에 세 개의 컴퓨팅 노드를 프로비저닝합니다.

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  - hostname: overcloud-novacompute-1
    name: node05
  - hostname: overcloud-novacompute-2
    name: node06
```

기본적으로 프로비저닝 프로세스에서 **overcloud-full** 이미지를 사용합니다. **instances** 매개변수에서 **image** 속성을 사용하여 사용자 지정 이미지를 정의할 수 있습니다.

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
    image:
      href: overcloud-custom
```

각 노드 항목의 수동 노드 정의를 방지하도록 기본 매개변수 값을 **defaults** 매개변수로 덮어쓸 수도 있습니다.

```
- name: Controller
  count: 3
  defaults:
    image:
      href: overcloud-custom
  instances:
  - hostname :overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
```

노드 정의 파일에서 사용할 수 있는 매개변수, 속성 및 값에 대한 자세한 내용은 베어 메탈 노드 프로비저닝 특성을 참조하십시오.

4. ~/**overcloud-baremetal-deploy.yaml** 파일을 지정하고 **--output** 옵션으로 출력 파일을 정의하여 프로비저닝 명령을 실행합니다.

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

프로비저닝 프로세스에서는 사용자가 **--output** 옵션에 지정하는 이름을 사용하여 **heat** 환경 파일을 생성합니다. 이 파일에는 노드 정의가 포함되어 있습니다. 오버클라우드를 배포할 때 배포 명령에 이 파일을 포함하십시오.

5. 개별 터미널에서 노드를 모니터링하여 제대로 프로비저닝되었는지 확인하십시오. 프로비저닝 프로세스에서 노드 상태를 **available**에서 **active**로 변경합니다.

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

metalsmith 도구를 사용하여 할당 및 **neutron** 포트를 비롯한 노드의 통합 보기를 얻을 수 있습니다.

```
(undercloud)$ metalsmith list
```

openstack baremetal allocation 명령을 사용하여 노드와 호스트 이름의 연관을 확인할 수도

있습니다.

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

노드가 제대로 프로비저닝되면 오버클라우드를 배포할 수 있습니다. 자세한 내용은 사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 설정을 참조하십시오.

8.4. 베어 메탈 노드 확장

기존 오버클라우드에서 베어 메탈 노드 개수를 늘리려면 ~/overcloud-baremetal-deploy.yaml 파일에서 노드 개수를 늘리고 오버클라우드를 다시 배포하십시오.

사전 요구 사항

- 성공적인 언더클라우드 설치 자세한 내용은 director 설치를 참조하십시오.
- 성공적인 오버클라우드 배포. 자세한 내용은 사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 설정을 참조하십시오.
- 프로비저닝 및 배포에 사용 가능하고 인트로스펙트된 베어 메탈 노드. 자세한 내용은 오버클라우드 노드 등록 및 베어 메탈 노드 하드웨어 인벤토리 생성을 참조하십시오.

절차

1. stackrc 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 베어 메탈 노드를 프로비저닝하는 데 사용한 ~/overcloud-baremetal-deploy.yaml 파일을 편집하고 확장할 역할의 count 매개변수를 늘립니다. 예를 들어 오버클라우드에 세 개의 컴퓨팅 노드가 있으면 다음 스니펫을 사용하여 컴퓨팅 노드 개수를 10으로 늘립니다.

```
- name: Controller
  count: 3
- name: Compute
  count: 10
```

instances 매개변수로 예측 가능한 노드 배치를 추가할 수 있습니다. 사용 가능한 매개변수

및 속성에 대한 자세한 내용은 베어 메탈 노드 프로비저닝 특성을 참조하십시오.

3. ~/**overcloud-baremetal-deploy.yaml** 파일을 지정하고 **--output** 옵션으로 출력 파일을 정의하여 프로비저닝 명령을 실행합니다.

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

4. **openstack baremetal node list** 명령을 사용하여 프로비저닝 진행 상태를 모니터링합니다.

5. 배포에 관련된 기타 환경 파일과 함께 프로비저닝 명령을 통해 생성하는 ~/**overcloud-baremetal-deployed.yaml** 파일을 비롯한 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
environment.yaml \
  -e ~/overcloud-baremetal-deployed.yaml \
  --deployed-server \
  --disable-validations \
  ...
```

8.5. 베어 메탈 노드 축소

~/**overcloud-baremetal-deploy.yaml** 파일의 스택에서 삭제할 노드를 태깅하고, 오버클라우드를 다시 배포한 다음, **--baremetal-deployment** 옵션을 사용하여 **openstack overcloud node delete** 명령에 이 파일을 포함합니다.

사전 요구 사항

- 성공적인 언더클라우드 설치 자세한 내용은 **4장. *언더클라우드에 director 설치*** 의 내용을 참조하십시오.
- 성공적인 오버클라우드 배포. 자세한 내용은 **9장. *사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 설정***의 내용을 참조하십시오.
- 스택에서 삭제할 하나 이상의 베어 메탈 노드.

절차

1. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 베어 메탈 노드를 프로비저닝하는 데 사용한 **~/overcloud-baremetal-deploy.yaml** 파일을 편집하고 축소하는 역할의 **count** 매개변수를 줄입니다. 스택에서 삭제할 각 노드의 다음 속성도 정의해야 합니다.

- 노드의 이름.
- 노드와 연관된 호스트 이름.
- **provisioned: false** 속성.

예를 들어 스택에서 **overcloud-controller-1** 노드를 삭제하려면 **~/overcloud-baremetal-deploy.yaml** 파일에 스니펫을 포함합니다.

```
- name: Controller
  count: 2
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
    # Removed from cluster due to disk failure
    provisioned: false
  - hostname: overcloud-controller-2
    name: node02
```

3. **~/overcloud-baremetal-deploy.yaml** 파일을 지정하고 **--output** 옵션으로 출력 파일을 정의하여 프로비저닝 명령을 실행합니다.

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

4. 배포에 관련된 기타 환경 파일과 함께 프로비저닝 명령을 통해 생성되는 **~/overcloud-baremetal-deployed.yaml** 파일을 포함하여 오버클라우드를 다시 배포합니다.

```
(undercloud)$ openstack overcloud deploy \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
environment.yaml \
  -e ~/overcloud-baremetal-deployed.yaml \
  --deployed-server \
  --disable-validations \
  ...
```

오버클라우드를 다시 배포하고 나면 provisioned: false 속성으로 정의한 노드가 더 이상 스택에 존재하지 않습니다. 그러나 이 노드는 여전히 프로비저닝된 상태로 실행 중입니다.

참고

스택에서 임시로 노드를 삭제하려면 provisioned: false 속성을 사용하여 오버클라우드를 배포한 다음 provisioned: true 속성으로 오버클라우드를 다시 배포하여 노드를 스택에 반환해야 합니다.

5. **--baremetal-deployment 옵션으로 ~/overcloud-baremetal-deploy.yaml 파일을 지정하여 openstack overcloud node delete 명령을 실행합니다.**

```
(undercloud)$ openstack overcloud node delete \
--stack stack \
--baremetal-deployment ~/overcloud-baremetal-deploy.yaml
```

참고

스택에서 삭제할 노드는 openstack overcloud node delete 명령에서 명령 인수로 포함하지 마십시오.

8.6. 베어 메탈 노드 프로비저닝 속성

다음 표에서는 openstack baremetal node provision 명령을 사용하여 베어 메탈 노드를 프로비저닝할 사용할 수 있는 매개변수, 속성 및 값을 설명합니다.

표 8.1. 역할 매개변수

매개변수	값
name	역할 이름 (필수)

매개변수	값
count	이 역할에 사용하도록 프로비저닝할 노드 수입니다. 기본값은 **1**입니다.
defaults	**instances** 항목 속성의 기본값 사전.**instances** 항목 속성을 통해 **defaults** 매개변수에서 지정하는 기본값을 덮어씁니다.
instances	특정 노드의 속성을 지정하는 데 사용할 수 있는 값의 사전입니다. **instances** 매개변수에서 지원되는 속성에 대한 자세한 내용은 표 8.2. "**instances** 및 **defaults** 매개변수"을 참조하십시오. 이 목록의 길이는 **count** 매개변수 값보다 크지 않아야 합니다.
hostname_format	이 역할의 기본 호스트 이름 형식을 덮어씁니다. 기본 형식에서 소문자 역할 이름을 사용합니다. 예를 들어 제어기 역할의 기본 형식은 **%stackname%-controller-%index%**입니다. 컴퓨팅 노드만 역할 이름 규칙을 따르지 않습니다. 컴퓨팅 노드의 기본 형식은 **%stackname%-novacompute-%index%**입니다.

예제 구문

다음 예에서 name은 노드의 논리 이름을 참조하고 hostname은 오버클라우드 스택 이름, 역할 및 증분 색인에서 파생되는 생성된 호스트 이름을 나타냅니다. 모든 컨트롤러 서버는 기본 사용자 지정 이미지 overcloud-full-custom을 사용하고 예측 가능한 노드에 있습니다. 컴퓨팅 서버 중 하나가 사용자 지정 호스트 이름 overcloud-compute-special을 사용하여 node04에 예측에 따라 배치되고, 다른 99개의 컴퓨팅 서버는 사용 가능한 노드 풀에서 자동으로 노드에 할당됩니다.

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 100
  instances:
  - hostname: overcloud-compute-special
    name: node04
```

표 8.2. instances 및 defaults 매개변수

매개변수	값
hostname	**hostname_format** 패턴을 사용하여 호스트 이름을 컴파일하면 다른 속성이 이 호스트 이름에 할당된 노드에 적용됩니다. 그렇지 않으면 이 노드의 사용자 정의 호스트 이름을 사용할 수 있습니다.
name	프로비저닝할 노드의 이름입니다.
image	노드에 프로비저닝할 이미지의 세부 정보입니다. **image** 매개변수에서 지원되는 속성에 대한 자세한 내용은 표 8.3. "**image** 매개변수"을 참조하십시오.
capabilities	노드 기능과 일치시킬 선택 기준입니다.
nics	요청된 NIC를 표시하는 사전 목록입니다. **nics** 매개변수에서 지원되는 속성에 대한 자세한 내용은 표 8.4. "**nic** 매개변수"을 참조하십시오.
profile	고급 프로필 일치를 사용할 때 선택 기준입니다.
provisioned	이 노드가 프로비저닝되었는지 아니면 프로비저닝 해제되었는지 판별하는 부울 값입니다. 기본값은 **true**입니다. **false**를 사용하여 노드의 프로비저닝을 해제합니다. 자세한 내용은 베어 메탈 노드 스케일링을 참조하십시오.
resource_class	노드의 리소스 클래스를 조합할 때의 선택 기준입니다. 기본값은 **baremetal**입니다.
root_size_gb	루트 파티션의 크기(GiB)입니다. 기본값은 **49**입니다.
swap_size_mb	스왑 파티션의 크기(MiB)입니다.
traits	노드 특성을 조합할 때 선택 기준인 특성 목록입니다.

예제 구문

다음 예에서 모든 제어 서버는 사용자 지정 기본 오버클라우드 이미지 overcloud-full-custom을 사용합니다. 컨트롤러 서버 overcloud-controller-0은 예측에 따라 node00에 배치되며 사용자 지정 루트 파티션 및 스왑 파티션 크기가 있습니다. 다른 두 컨트롤러 서버는 사용 가능한 노드 풀에서 자동으로 할당된 노드에 있으며, 기본 루트 파티션과 스왑 파티션 크기가 있습니다.

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  instances:
```

```
- hostname: overcloud-controller-0
  name: node00
  root_size_gb: 140
  swap_size_mb: 600
```

표 8.3. image 매개변수

매개변수	값
href	Glance 이미지 참조 또는 루트 파티션 또는 전체 디스크 이미지의 URL입니다. 지원되는 URL 스키마는 **file://**, **http://** 및 **https://**입니다. 값이 올바른 URL이 아니면 이 값은 올바른 Glance 이미지 참조여야 합니다.
checksum	href가 URL이면 이 값은 루트 파티션 또는 전체 디스크 이미지의 SHA512 체크섬이어야 합니다.
kernel	커널 이미지의 Glance 이미지 참조 또는 URL입니다. 파티션 이미지에만 이 속성을 사용합니다.
ramdisk	Ramdisk 이미지의 Glance 이미지 참조 또는 URL입니다. 파티션 이미지에만 이 속성을 사용합니다.

예제 구문

다음 예에서 세 개의 컨트롤러 서버는 모두 사용 가능한 노드 풀에서 자동으로 할당된 노드에 있습니다. 이 환경의 모든 컨트롤러 서버에서는 기본 사용자 정의 이미지 overcloud-full-custom을 사용합니다.

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
      checksum: 1582054665
      kernel: file:///var/lib/ironic/images/overcloud-full-custom.vmlinuz
      ramdisk: file:///var/lib/ironic/images/overcloud-full-custom.initrd
```

표 8.4. nic 매개변수

매개변수	값
fixed_ip	이 NIC에 사용할 특정 IP 주소입니다.
network	이 NIC의 포트를 생성할 neutron 네트워크입니다.
subnet	이 NIC의 포트를 생성할 neutron 서브넷입니다.
port	새 포트를 생성하는 대신 사용할 기존 Neutron 포트입니다.

예제 구문

다음 예에서 세 개의 컨트롤러 서버는 모두 사용 가능한 노드 풀에서 자동으로 할당된 노드에 있습니다. 이 환경의 모든 컨트롤러 서버에는 기본 사용자 정의 이미지 overcloud-full-custom을 사용하며 다음과 같은 특정 네트워킹 요구 사항이 있습니다.

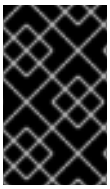
```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
      nics:
        network: custom-network
        subnet: custom-subnet
```

9장. 사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 설정

이 장에는 사전 프로비저닝된 노드가 있는 **RHOSP(Red Hat OpenStack Platform)** 환경을 설정하는 데 사용할 수 있는 기본 설정 절차가 포함됩니다. 이 시나리오는 여러 방식에서 표준 오버클라우드 생성 시나리오와 다릅니다.

- 외부 툴을 사용하여 노드를 프로비저닝하고 **director**가 오버클라우드 구성만 제어하도록 할 수 있습니다.
- **director** 프로비저닝 방법에 의존하지 않고 노드를 사용할 수 있습니다. 이 시나리오는 전원 관리 제어 없이 오버클라우드를 생성하거나 **DHCP/PXE** 부팅 제한이 있는 네트워크를 사용하려는 경우에 유용합니다.
- **director**에서 노드 관리에 **OpenStack Compute(nova), OpenStack Bare Metal(ironic)** 또는 **OpenStack Image(glance)**를 사용하지 않습니다.
- 사전 프로비저닝된 노드에서는 **QCOW2 overcloud-full** 이미지를 사용하지 않는 사용자 지정 파티션 레이아웃을 사용할 수 있습니다.

이 시나리오에는 사용자 지정 기능이 없는 기본 설정만 포함됩니다. 하지만 **Advanced Overcloud Customization** 가이드의 지침을 사용하여 이 기본 오버클라우드에 고급 구성 옵션을 추가하고 사양에 맞게 사용자 지정할 수 있습니다.



중요

사전 프로비저닝된 노드와 **director** 프로비저닝된 노드를 결합할 수 없습니다.

9.1. 사전 프로비저닝된 노드 요구 사항

사전 프로비저닝된 노드가 있는 오버클라우드 배포를 시작하기 전에 사용자 환경에 다음 구성이 있는지 확인합니다.

- **4장. *언더클라우드에 director 설치***에서 생성한 **director** 노드
- 노드에 사용할 베어 메탈 시스템 세트. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다. 해당 머신은 각 노드 유형에 설정된 요구 사항을 준수해야 합니다. 이러한 노드에는

Red Hat Enterprise Linux 8.4를 호스트 운영 체제로 설치해야 하며, 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 계층 3(L3) 라우팅 가능한 네트워크 연결입니다. 이 시나리오의 예에서는 다음 IP 주소 할당을 사용합니다.

    표 9.1. 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 시나리오의 요구 사항에 대한 자세한 내용은 9.6절. “사전 프로비저닝된 노드에 별도의 네트워크 사용”을 참조하십시오.

- 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 haclient의 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.

9.2. 사전 프로비저닝된 노드에서 사용자 생성

사전 프로비저닝된 노드를 사용하여 오버클라우드를 구성하는 경우 **director**는 오버클라우드 노드에 대한 **SSH** 액세스 권한이 필요합니다. 사전 프로비저닝된 노드에서 **SSH** 키 인증을 사용하여 사용자를 생성하고 해당 사용자에 대해 암호 없이 **sudo** 액세스를 구성해야 합니다. 사전 프로비저닝된 노드에 사용자를 생성한 후 **openstack overcloud deploy** 명령과 함께 **--overcloud-ssh-user** 및 **--overcloud-ssh-key** 옵션을 사용하여 사전 프로비저닝된 노드가 있는 오버클라우드를 생성할 수 있습니다.

기본적으로 오버클라우드 **SSH** 사용자 및 오버클라우드 **SSH** 키 값은 **stack** 사용자와 **~/.ssh/id_rsa**입니다. **stack** 사용자를 생성하려면 다음 단계를 완료합니다.

절차

1. 각 오버클라우드 노드에서 **stack** 사용자를 생성하고 암호를 설정합니다. 예를 들어 컨트롤러 노드에서 다음 명령을 실행합니다.

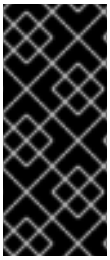
```
[root@controller-0 ~]# useradd stack
[root@controller-0 ~]# passwd stack  # specify a password
```

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 구성한 후 **director** 노드에서 각 오버클라우드 노드로 **stack** 사용자의 공용 **SSH** 키를 복사합니다. 예를 들어 **director**의 공용 **SSH** 키를 컨트롤러 노드에 복사하려면 다음 명령을 실행합니다.

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



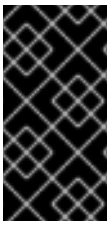
중요

SSH 키를 복사하려면 각 오버클라우드 노드의 **SSH** 구성에 **PasswordAuthentication Yes**를 일시적으로 설정해야 할 수 있습니다. **SSH** 키를 복사한 후 **PasswordAuthentication No**를 설정하고 **SSH** 키를 사용하여 나중에 인증합니다.

9.3. 사전 프로비저닝된 노드의 운영 체제 등록

각 노드는 **Red Hat** 서브스크립션에 액세스할 수 있어야 합니다. 노드를 **Red Hat Content Delivery**

Network에 등록하려면 각 노드에서 다음 단계를 완료합니다.



중요

나열된 리포지토리만 활성화합니다. 추가 리포지토리를 사용하면 패키지와 소프트웨어가 충돌할 수 있습니다. 추가 리포지토리를 활성화하지 마십시오.

절차

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

```
[heat-admin@controller-0 ~]$ sudo subscription-manager register
```

2. **Red Hat OpenStack Platform 16.2**에 대한 인타이틀먼트 풀을 검색합니다.

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

3. 이전 단계에 있는 풀 **ID**를 사용하여 **Red Hat OpenStack Platform 16** 인타이틀먼트를 연결합니다.

```
[heat-admin@controller-0 ~]$ sudo subscription-manager attach --pool=pool_id
```

4. 모든 기본 리포지토리를 비활성화합니다.

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --disable=*
```

5. 필수 **Red Hat Enterprise Linux** 리포지토리를 활성화합니다.

   a. **x86_64** 시스템의 경우 다음 명령을 실행합니다.

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-16.2-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms
```

b.

POWER 시스템의 경우 다음 명령을 실행합니다.

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-ppc64le-baseos-rpms --enable=rhel-8-for-ppc64le-appstream-rpms --enable=rhel-8-for-ppc64le-highavailability-rpms --enable=ansible-2.8-for-rhel-8-ppc64le-rpms --enable=openstack-16-for-rhel-8-ppc64le-rpms --enable=fast-datapath-for-rhel-8-ppc64le-rpms
```

6.

container-tools 리포지토리 모듈을 버전 **3.0**으로 설정합니다.

```
[heat-admin@controller-0 ~]$ sudo dnf module disable -y container-tools:rhel8
[heat-admin@controller-0 ~]$ sudo dnf module enable -y container-tools:3.0
```

7.

오버클라우드에서 **Ceph Storage** 노드를 사용하는 경우 관련 **Ceph Storage** 리포지토리를 활성화합니다.

```
[heat-admin@cephstorage-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms --enable=rhel-8-for-x86_64-appstream-rpms --enable=rhel-8-for-x86_64-highavailability-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms
```

8.

dnf 업데이트를 실행하기 전에 각 노드를 **Red Hat Enterprise Linux 8.4**에 고정합니다.

```
[heat-admin@controller-0 ~]$ sudo subscription-manager release --set=8.4
```

9.

시스템을 업데이트하여 기본 시스템 패키지를 최신 상태로 유지합니다.

```
[heat-admin@controller-0 ~]$ sudo dnf update -y
[heat-admin@controller-0 ~]$ sudo reboot
```

이제 노드에서 오버클라우드를 사용할 준비가 되었습니다.

9.4. DIRECTOR로의 SSL/TLS 액세스 설정

director가 **SSL/TLS**를 사용하는 경우 사전 프로비저닝된 노드에 **director**의 **SSL/TLS** 인증서에 서명하는 데 사용된 인증 기관 파일이 필요합니다. 고유한 인증 기관을 사용하는 경우 각 오버클라우드 노드에서 다음 작업을 수행합니다.

절차

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
```

이 단계를 수행하면 오버클라우드 노드에서 **SSL/TLS**를 통해 **director**의 공용 **API**에 액세스할 수 있습니다.

9.5. 컨트롤 플레인 네트워킹 구성

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

- **director**의 컨트롤 플레인 네트워크. **undercloud.conf** 파일의 **network_cidr** 매개변수로 정의된 서브넷입니다. 오버클라우드 노드에는 이 서브넷에 대한 직접 액세스 권한이나 서브넷으로 라우팅 가능한 액세스 권한이 필요합니다.

- **director**의 공용 **API** 엔드포인트. **undercloud.conf** 파일의 **undercloud_public_host** 매개변수로 지정합니다. 이 옵션은 컨트롤 플레인에 대한 **L3** 경로가 없거나, **SSL/TLS** 통신을 사용하려는 경우에 사용할 수 있습니다. 공용 **API** 엔드포인트를 사용하도록 오버클라우드 노드를 구성하는 방법에 대한 자세한 내용은 **9.6절. "사전 프로비저닝된 노드에 별도의 네트워크 사용"**을 참조하십시오.

director는 컨트롤 플레인 네트워크를 사용하여 표준 오버클라우드를 관리하고 설정합니다. 노드가 사전 프로비저닝된 오버클라우드의 경우 **director**와 사전 프로비저닝된 노드 간에 통신할 수 있도록 네트워크 구성을 일부 변경해야 할 수 있습니다.

네트워크 분리 사용

네트워크 분리를 사용하여 컨트롤 플레인을 비롯한 특정 네트워크를 사용하도록 서비스를 그룹화할 수 있습니다. **Advanced Overcloud Customization** 가이드에 여러 네트워크 분리 전략이 나와 있습니다. 컨트롤 플레인에서 노드의 특정 **IP** 주소를 정의할 수도 있습니다. 네트워크를 분리하고 예측 가능한 노드 배치 설정을 생성하는 방법에 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 **NIC**가 포함되지 않도록 합니다. 이러한 템플릿은 **NIC**를 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 **NIC**를 수동으로 설정해야 합니다. **director**의 프로비저닝 네트워크를 컨트롤 플레인으로 사용하는 경우 선택한 오버클라우드 **IP** 주소가 프로비저닝(**dhcp_start** 및 **dhcp_end**)과 인트로스펙션(**inspection_iprange**)에 대한 **DHCP** 범위를 벗어나는지 확인합니다.

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

예를 들어 다음 **IP**가 할당된 **ctlplane-assignments.yaml** 환경 파일을 사용하여 예측 가능한 **IP** 설정을 구현할 수 있습니다.

```
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: 192.168.24.0/24
      network:
        tags:
          192.168.24.0/24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.3
      subnets:
        - cidr: 192.168.24.0/24
```

```
  network:
    tags:
      - 192.168.24.0/24
```

이 예제에서 **OS::TripleO::DeployedServer::ControlPlanePort** 리소스는 매개변수 세트를 **director**에 전달하고, 사전 프로비저닝된 노드의 **IP** 할당을 정의합니다. **DeployedServerPortMap** 매개변수를 사용하여 각 오버클라우드 노드에 해당하는 **IP** 주소와 서브넷 **CIDR**을 정의합니다. 매핑은 다음 속성을 정의합니다.

1. 할당 이름 - **<node_hostname>-<network>** 포멧을 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 다음 부분에서는 생성된 환경 파일(**ctlplane-assignments.yaml**)을 **openstack overcloud deploy** 명령의 일부로 사용합니다.

9.6. 사전 프로비저닝된 노드에 별도의 네트워크 사용

기본적으로 **director**는 프로비저닝 네트워크를 오버클라우드 컨트롤 플레인으로 사용합니다. 하지만 이 네트워크가 분리되어 라우팅이 불가능한 경우 구성 중에 노드에서 **director** 내부 **API**와 통신할 수 없습니다. 이 경우 노드에 대해 별도의 네트워크를 정의하고, 공용 **API**로 **director**와 통신하도록 구성해야 할 수 있습니다.

이 시나리오에는 다음과 같은 여러 요구 사항이 있습니다.

- 오버클라우드 노드는 **9.5절. “컨트롤 플레인 네트워킹 구성”**의 기본 네트워크 설정을 수용해야 합니다.

- **director**에서 공용 **API** 엔드포인트 사용에 대해 **SSL/TLS**를 활성화해야 합니다. 자세한 내용

은 4.2절. “director 설정 매개변수” 및 20장. *사용자 지정 SSL/TLS 인증서 설정*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 9.2. 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.

IP 주소 할당

IP 할당 방법은 9.5절. “컨트롤 플레인 네트워킹 구성”과 비슷합니다. 하지만 컨트롤 플레인은 배포된 서버에서 라우팅되지 않으므로 DeployedServerPortMap 매개변수를 사용하여 컨트롤 플레인에 액세스할 가상 IP 주소를 포함하여 선택한 오버클라우드 노드 서브넷에서 IP 주소를 할당해야 합니다. 다음 예제는 9.5절. “컨트롤 플레인 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
  OS::TripleO::Network::Ports::OVNDBsVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml (1)

parameter_defaults:
  NeutronPublicInterface: eth1
```

```
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 및 **OVNDBsVipPort** 리소스는 **network/ports/noop.yaml** 에 매핑됩니다. 이 매핑은 기본 **Redis** 및 **OVNDBs VIP** 주소가 컨트롤 플레인에서 제공되기 때문에 필요합니다. 이 경우 **noop**를 사용하여 이 컨트롤 플레인 매핑을 비활성화합니다.

9.7. 사전 프로비저닝된 노드 호스트 이름 매핑

사전 프로비저닝된 노드를 구성할 때 **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 추가합니다. 다음 구문을 사용합니다.

```
parameter_defaults:
  HostnameMap:
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
```

[HEAT HOSTNAME]은 일반적으로 **[STACK NAME]-[ROLE]-[INDEX]** 의 표기법을 따릅니다.

```
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** 파일을 저장합니다.

9.8. 사전 프로비저닝된 노드에 대해 CEPH STORAGE 설정

사전 배포된 노드에 대해 **ceph-anible**을 설정하려면 언더클라우드 호스트에서 다음 단계를 완료합니다.

절차

1. 언더클라우드 호스트에서 **OVERCLOUD_HOSTS** 환경 변수를 생성하고, **Ceph** 클라이언트로 사용하려는 오버클라우드 호스트 **IP** 주소의 공백으로 구분된 목록으로 변수를 설정합니다.

```
export OVERCLOUD_HOSTS="192.168.1.8 192.168.1.42"
```

2. 기본 오버클라우드 계획 이름은 **overcloud**입니다. 다른 이름을 사용하는 경우 사용자 지정 이름을 저장할 환경 변수 **OVERCLOUD_PLAN**을 생성합니다.

```
export OVERCLOUD_PLAN="<custom-stack-name>"
```

- **<custom-stack-name>**을 스택 이름으로 바꿉니다.

3. **enable-ssh-admin.sh** 스크립트를 실행하여 **Ansible**이 **Ceph** 클라이언트를 설정하는 데 사용할 수 있는 오버클라우드 노드에 사용자를 설정합니다.

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

openstack overcloud deploy 명령을 실행하면 **Ansible**에서 **OVERCLOUD_HOSTS** 변수에 정의된 호스트를 **Ceph** 클라이언트로 설정합니다.

9.9. 사전 프로비저닝된 노드를 사용하여 오버클라우드 생성

오버클라우드 배포에서는 7.14절. “배포 명령”의 표준 **CLI** 메서드를 사용합니다. 사전 프로비저닝된 노

드의 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 비활성화하려면 이 옵션을 사용합니다. 해당 검증을 비활성화하지 않으면 배포에 실패합니다.

- **environments/deployed-server-environment.yaml -** 사전 프로비저닝된 인프라를 생성 및 구성하려면 이 환경 파일을 추가합니다. 이 환경 파일은 **OS::Nova::Server** 리소스를 **OS::Heat::DeployedServer** 리소스로 대체합니다.

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

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
  -e /home/stack/templates/hostname-map.yaml \
  --overcloud-ssh-user stack \
  --overcloud-ssh-key ~/.ssh/id_rsa \
  <OTHER OPTIONS>
```

--overcloud-ssh-user 및 **--overcloud-ssh-key** 옵션은 구성 단계 중 각 오버클라우드 노드에 **SSH** 사용, 초기 **tripleo-admin** 사용자 생성 및 **SSH** 키를 **/home/tripleo-admin/.ssh/authorized_keys**에 삽입하는 데 사용됩니다. **SSH** 키를 삽입하려면 **--overcloud-ssh-user** 및 **--overcloud-ssh-key(~/.ssh/id_rsa**로 기본 설정됨**)**을 사용하여 초기 **SSH** 연결의 인증서를 지정합니다. **--overcloud-ssh-key** 옵션으로 지정하는 개인 키로 공개를 제한하기 위해 **director**는 **heat** 또는 **Workflow** 서비스**(mistral)**와 같은 **API** 서비스에 이 키를 전달하지 않으며**, director openstack overcloud deploy** 명령만 이 키를 사용하여 **tripleo-admin** 사용자의 액세스를 활성화합니다.

9.10. 오버클라우드 배포 출력

오버클라우드 생성이 완료되면 **director**가 오버클라우드를 설정하기 위해 실행한 **Ansible** 플레이 개요를 표시합니다.

```
PLAY RECAP *********************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
===============================================================================
```

director는 또한 오버클라우드 액세스에 필요한 세부 정보도 제공합니다.

```
Ansible passed.
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
```

9.11. 오버클라우드 액세스

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

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

이 명령은 언더클라우드 **CLI**에서 오버클라우드와 상호 작용하는 데 필요한 환경 변수를 로드합니다. 명령 프롬프트가 다음과 같이 변경됩니다.

```
(overcloud) $
```

언더클라우드와 상호 작용으로 돌아가려면 다음 명령을 실행합니다.

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

9.12. 사전 프로비저닝된 노드 확장

사전 프로비저닝된 노드를 확장하는 프로세스는 *16장. 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 **director** 노드 수에 해당하도록 오케스트레이션 에이전트를 설정해야 합니다.

오버클라우드 노드를 확장하려면 다음 작업을 수행합니다.

1. 9.1절. "사전 프로비저닝된 노드 요구 사항"에 따라 사전 프로비저닝된 새 노드를 준비합니다.
2. 노드를 확장합니다. 자세한 내용은 16장. *오버클라우드 노드 확장*의 내용을 참조하십시오.
3. 배포 명령을 실행한 후 director가 새 노드 리소스를 생성하고 설정을 시작할 때까지 기다립니다.

사전 프로비저닝된 노드 축소

사전 프로비저닝된 노드가 있는 오버클라우드를 축소하는 경우 16장. *오버클라우드 노드 확장*의 축소 지침을 따릅니다.

스케일 다운 작업에서는 OSP 프로비저닝 또는 사전 프로비저닝된 노드 모두에 호스트 이름을 사용할 수 있습니다. OSP 프로비저닝된 노드의 UUID도 사용할 수 있습니다. 그러나 사전 테스트된 노드의 UUID는 없으므로 항상 호스트 이름을 사용합니다. 호스트 이름 또는 UUID 값을 **openstack overcloud node delete** 명령에 전달합니다.

절차

1. 제거할 노드의 이름을 확인합니다.

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

2. **stack_name** 열의 해당 노드 이름을 **openstack overcloud node delete** 명령에 전달합니다.

```
$ openstack overcloud node delete --stack <overcloud> <stack>
```

- **<overcloud>**를 오버클라우드 스택의 이름 또는 **UUID**로 바꿉니다.
- **& lt;stack_name >**을 삭제하려는 노드 이름으로 바꿉니다. **openstack overcloud node delete** 명령에 여러 노드 이름을 포함할 수 있습니다.

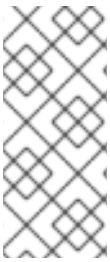
3. **openstack overcloud node delete** 명령 실행이 완전히 종료되었는지 확인합니다.

```
$ openstack stack list
```

삭제 작업이 완료되면 **overcloud** 스택의 상태가 **UPDATE_COMPLETE**로 표시됩니다.

스택에서 오버클라우드 노드를 삭제한 후 이러한 노드의 전원을 끕니다. 표준 배포에서는 **director**의 베어 메탈 서비스가 이 기능을 제어합니다. 하지만 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.

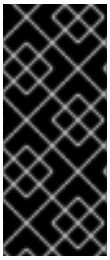
사전 프로비저닝된 오버클라우드 삭제

사전 프로비저닝된 노드를 사용하는 전체 오버클라우드를 삭제하려면 표준 오버클라우드 삭제 절차는 12.6절. “오버클라우드 삭제”에서 참조하십시오. 오버클라우드 삭제 후에 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.

10장. 여러 오버클라우드 배포



중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

단일 언더클라우드 노드를 사용하여 여러 오버클라우드를 배포 및 관리할 수 있습니다. 각 오버클라우드는 스택 리소스를 공유하지 않는 고유한 **heat** 스택입니다. 이 기능은 언더클라우드와 오버클라우드의 비율이 **1:1**이라서 생성되는 오버헤드를 관리할 수 없는 환경에서 유용할 수 있습니다. 예를 들어 **Edge**, 다중 사이트, 다중 제품 환경 등이 있습니다.

여러 오버클라우드 배포의 오버클라우드 환경은 완전히 분리되어 있으며, **source** 명령을 사용하여 환경 간에 전환할 수 있습니다. 각 오버클라우드에는 배포 프로세스에서 생성한 고유한 인증 정보 파일이 있습니다. 오버클라우드와 상호 작용하려면 적절한 인증 정보 파일을 소싱해야 합니다.

베어 메탈 프로비저닝에 **Bare Metal Provisioning** 서비스(**ironic**)를 사용하는 경우 모든 오버클라우드가 동일한 프로비저닝 네트워크에 있어야 합니다. 동일한 프로비저닝 네트워크를 사용할 수 없는 경우 배포된 서버 방법을 사용하여 라우팅된 네트워크로 여러 오버클라우드를 배포할 수 있습니다. 이 시나리오에서는 **HostnameMap** 매개변수의 값이 각 오버클라우드의 스택 이름과 일치하는지 확인해야 합니다.

단일 언더클라우드에 여러 오버클라우드를 배포하려면 다음 작업을 수행해야 합니다.

1. 언더클라우드를 배포합니다. 자세한 내용은 **Part I. Director 설치 및 설정**.
2. 첫 번째 오버클라우드를 배포합니다. 자세한 내용은 **Part II**를 참조하십시오. 기본 오버클라우드 배포.
3. 새 오버클라우드에 사용할 새 환경 파일 세트를 생성하고 배포 명령에 새 설정 파일 및 새 스택 이름과 함께 코어 **heat** 템플릿을 지정하여 추가 오버클라우드를 배포합니다.

10.1. 추가 오버클라우드 배포

단일 언더클라우드에 두 개 이상의 오버클라우드를 배포할 수 있습니다. 다음 절차에서는 기존 오버클라우드인 **overcloud-one** 이 있는 기존 **RHOSP(Red Hat OpenStack Platform)** 배포에서 새 오버클라우

드 **overcloud-two** 를 생성하고 배포하는 방법을 설명합니다.

사전 요구 사항

- 언더클라우드.
- 하나 이상의 오버클라우드.
- 추가 오버클라우드에 사용할 수 있는 노드입니다.
- 결과 스택에서 각 오버클라우드에 고유한 네트워크가 있도록 추가 오버클라우드에 사용할 사용자 지정 네트워크

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 배포하려는 추가 오버클라우드에 사용할 새 디렉터리를 생성합니다.

```
(undercloud)$ mkdir ~/overcloud-two
```

4. 기존 오버클라우드의 **network_data.yaml** 파일을 추가 오버클라우드의 새 디렉터리로 복사합니다.

```
(undercloud)$ cp network_data.yaml ~/overcloud-two/network_data.yaml
```

5. **~/overcloud-two/network_data.yaml** 파일을 열고 **name_lower** 를 추가 오버클라우드 네트워크의 고유한 이름으로 업데이트합니다.

```
- name: InternalApi
  name_lower: internal_api_cloud_2
  ...
```

6. **service_net_map_replace** 이 없으면 값을 추가 오버클라우드 네트워크의 기본값으로 설정합니다.

```
- name: InternalApi
  name_lower: internal_api_cloud_2
  service_net_map_replace: internal_api
```

7. 추가 오버클라우드에서 각 서브넷의 **VLAN ID**를 지정합니다.

```
- name: InternalApi
  ...
  vip: true
  vlan: 21
  ip_subnet: '172.21.0.0/24'
  allocation_pools: [{'start': '172.21.0.4', 'end': '172.21.0.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2001::10', 'end':
'fd00:fd00:fd00:2001:ffff:ffff:ffff:fffe'}]
  mtu: 1500
- name: Storage
  ...
```

8. **overcloud-two** 외부 네트워크의 게이트웨이 **IP** 주소를 지정합니다.

```
- name: External
  ...
  gateway_ip: <ip_address>
  ...
```

- **& lt;ip_address >**를 **overcloud-two** 외부 네트워크의 게이트웨이 **IP** 주소로 바꿉니다 **(예: 10.0.10.1).**

9. **/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml** 파일에 제공된 기본 분리된 네트워크 구성을 덮어쓰는 추가 오버클라우드의 네트워크 구성 파일을 만듭니다**(예: network_overrides.yaml).**

10. **~/overcloud-two/network_overrides.yaml** 파일을 열고 **overcloud-two DNS** 서버의 **IP** 주소를 추가합니다.

```
parameter_defaults:
  ...
  DnsServers:
```

```
    - <ip_address>
  ...
```

- & lt;ip_address >를 overcloud-two DNS 서버의 IP 주소로 바꿉니다(예: 10.0.10. 2).

11. 배포에서 예측 가능한 IP 주소를 사용하는 경우 새 네트워크 IP 주소 매핑 파일 ips-from-pool- overcloud-two.yaml에서 overcloud-two 노드의 IP 주소를 구성합니다.

```
parameter_defaults:
  ControllerIPs:
  ...
    internal_api_cloud_2:
    - 192.168.1.10
    - 192.168.1.11
    - 192.168.1.12
    ...
    external_cloud_2:
    - 10.0.1.41
    ...
```

12. 다른 환경 파일과 함께 overcloud-two 환경 파일을 스택에 추가하고 추가 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  --stack overcloud-two \
  -n ~/overcloud-two/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml
\
  -e ~/overcloud-two/network_overrides.yaml \
  -e [your environment files] \
  ...
```

배포 프로세스는 overcloud-two 와 상호 작용하고 관리하기 위해 overcloud-tworc 를 생성합니다.

13. 추가 오버클라우드와 상호 작용하려면 오버클라우드 인증 정보 파일을 소싱합니다.

```
$ source overcloud-tworc
```

10.2. 여러 오버클라우드 관리

배포하는 각 오버클라우드는 동일한 코어 **heat** 템플릿 세트인 **/usr/share/openstack-tripleo-heat-templates**을 사용합니다. 비표준 코어 템플릿 세트를 사용하면 업데이트 및 업그레이드 관련 문제가 발생할 수 있으므로 이러한 템플릿을 수정하거나 복제하지 않는 것이 좋습니다.

대신 여러 오버클라우드를 배포하거나 유지보수할 때 관리하기 쉽도록 각 클라우드와 관련된 개별 환경 파일 디렉터리를 생성합니다. 각 클라우드에 대해 배포 명령을 실행하는 경우 별도로 생성한 클라우드별 환경 파일과 함께 코어 **heat** 템플릿을 포함합니다. 예를 들어 언더클라우드와 오버클라우드 **2**개에 대해 다음 디렉터리를 생성합니다.

~stack/undercloud

언더클라우드와 관련된 환경 파일을 포함합니다.

~stack/overcloud-one

첫 번째 오버클라우드와 관련된 환경 파일을 포함합니다.

~stack/overcloud-two

두 번째 오버클라우드와 관련된 환경 파일을 포함합니다.

overcloud-one 또는 **overcloud-two** 중 하나를 배포하거나 재배포하는 경우 **--templates** 옵션과 함께 배포 명령에 코어 **heat** 템플릿을 추가하고 클라우드별 환경 파일 디렉터리의 추가 환경 파일을 지정합니다.

또는 버전 제어 시스템에 리포지토리를 생성하고 각 배포에 대해 분기를 사용합니다. 자세한 내용은 ***Advanced Overcloud Customization*** *가이드*의 **Using Customized Core Heat Templates** 섹션을 참조하십시오.

사용 가능한 오버클라우드 계획 목록을 보려면 다음 명령을 사용합니다.

```
$ openstack overcloud plan list
```

현재 배포된 오버클라우드 목록을 보려면 다음 명령을 사용합니다.

```
$ openstack stack list
```

11장. 오버클라우드 설치 후 작업 수행

이 장에서는 오버클라우드 생성 후 즉시 수행해야 하는 작업에 대해 설명합니다. 이러한 작업을 수행하면 오버클라우드를 사용할 준비가 완료됩니다.

11.1. 오버클라우드 배포 상태 확인

오버클라우드 배포 상태를 확인하려면 **openstack overcloud status** 명령을 사용합니다. 이 명령을 수행하면 모든 배포 단계의 결과가 반환됩니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 배포 상태 명령을 실행합니다.

```
$ openstack overcloud status
```

이 명령의 출력에는 오버클라우드 상태가 표시됩니다.

```
+-----------+---------------------+---------------------+-------------------+
| Plan Name |       Created       |       Updated       | Deployment Status |
+-----------+---------------------+---------------------+-------------------+
| overcloud | 2018-05-03 21:24:50 | 2018-05-03 21:27:59 |   DEPLOY_SUCCESS  |
+-----------+---------------------+---------------------+-------------------+
```

오버클라우드에서 다른 이름을 사용하는 경우 **--plan** 인수를 사용하여 다른 이름으로 오버클라우드를 선택합니다.

```
$ openstack overcloud status --plan my-deployment
```

11.2. 기본 오버클라우드 플레이버 생성

이 가이드의 검증 단계에서는 설치에 플레이버가 포함된 것으로 간주합니다. 플레이버를 하나 이상 생성하지 않은 경우 다양한 스토리지 및 처리 기능이 있는 기본 플레이버 세트를 생성하려면 다음 단계를 완료합니다.

절차

1. **overcloudrc** 파일을 소싱합니다.

```
$ source ~/overcloudrc
```

2. **openstack flavor create** 명령을 실행하여 플레이버를 생성합니다. 다음 옵션을 사용하여 각 플레이버에 대한 하드웨어 요구 사항을 지정합니다.

--disk

가상 머신 볼륨의 하드 디스크 공간을 정의합니다.

--ram

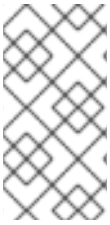
가상 머신에 필요한 **RAM**을 정의합니다.

--vcpus

가상 머신의 가상 **CPU** 수량을 정의합니다.

3. 기본 오버클라우드 플레이버를 생성의 예는 다음과 같습니다.

```
$ 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
```



참고

openstack flavor create 명령을 자세히 알아보려면 **$ openstack flavor create --help**를 사용합니다.

11.3. 기본 테넌트 네트워크 생성

오버클라우드에는 가상 머신이 내부적으로 통신할 수 있도록 기본 테넌트 네트워크가 필요합니다.

절차

1. **overcloudrc** 파일을 소싱합니다.

```
$ source ~/overcloudrc
```

2. 기본 테넌트 네트워크를 생성합니다.

```
(overcloud) $ openstack network create default
```

3. 네트워크에 서브넷을 생성합니다.

```
(overcloud) $ openstack subnet create default --network default --gateway 172.20.1.1 --
subnet-range 172.20.0.0/16
```

4. 생성된 네트워크를 확인합니다.

```
(overcloud) $ openstack network list
+----------------------+-------------+--------------------------------------+
| id                   | name        | subnets                              |
+----------------------+-------------+--------------------------------------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
+----------------------+-------------+--------------------------------------+
```

이 명령을 수행하면 **default**라는 기본 네트워킹 서비스**(neutron)** 네트워크가 생성됩니다. 오버클라우드에서 내부 **DHCP** 메커니즘을 사용하여 이 네트워크의 **IP** 주소를 가상 머신에 자동으로 할당합니다.

11.4. 기본 유동 IP 네트워크 생성

오버클라우드 외부에서 가상 머신에 액세스하려면 가상 머신에 유동 **IP** 주소를 제공하는 외부 네트워크를 구성해야 합니다.

이 절차에는 두 가지 예제가 있습니다. 환경에 가장 적합한 예제를 사용합니다.

- 기본 **VLAN(**플랫 네트워크**)**
- 기본이 아닌 **VLAN (VLAN** 네트워크**)**

두 예제는 모두 이름이 **public**인 네트워크를 생성합니다. 오버클라우드에는 기본 유동 **IP** 풀에 이러한 특정 이름이 필요합니다. 이 이름은 **11.7절. "오버클라우드 검증"**의 검증 테스트에서도 중요합니다.

기본적으로 **Openstack Networking(neutron)**은 **datacentre**라는 물리적 네트워크 이름을 호스트 노드의 **br-ex** 브릿지에 매핑합니다. **public** 오버클라우드 네트워크를 물리 네트워크 **datacentre**에 연결하면 **br-ex** 브릿지를 통한 게이트웨이가 제공됩니다.

사전 요구 사항

- 유동 **IP** 네트워크의 전용 인터페이스 또는 기본 **VLAN**

절차

1. **overcloudrc** 파일을 소싱합니다.

```
$ source ~/overcloudrc
```

2. **public** 네트워크를 생성합니다.

- 기본 **VLAN** 연결에 사용할 **flat** 네트워크를 생성합니다.

```
(overcloud) $ openstack network create public --external --provider-network-type flat --
provider-physical-network datacentre
```

- 기본이 아닌 **VLAN** 연결에 사용할 **vlan** 네트워크를 생성합니다.

```
(overcloud) $ openstack network create public --external --provider-network-type vlan --
provider-physical-network datacentre --provider-segment 201
```

--provider-segment 옵션을 사용하여 사용하려는 **VLAN**을 정의합니다. 이 예제에서 **VLAN**은 **201**입니다.

3. 유동 **IP** 주소에 대한 할당 풀을 사용하여 서브넷을 생성합니다. 이 예제에서 **IP** 범위는 **10.1.1.51~10.1.1.250**입니다.

```
(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 범위가 외부 네트워크의 다른 IP 주소와 충돌하지 않는지 확인합니다.

11.5. 기본 공급자 네트워크 생성

개인 테넌트 네트워크에서 외부 인프라 네트워크로 트래픽을 라우팅하는 다른 외부 네트워크 연결 유형의 공급자 네트워크입니다. 이 공급자 네트워크는 유동 IP 네트워크와 비슷하지만 공급자 네트워크는 논리 라우터를 사용하여 개인 네트워크를 공급자 네트워크에 연결합니다.

이 절차에는 두 가지 예제가 있습니다. 환경에 가장 적합한 예제를 사용합니다.

- 기본 VLAN(플랫 네트워크)
- 기본이 아닌 VLAN (VLAN 네트워크)

기본적으로 Openstack Networking(neutron)은 datacentre라는 물리적 네트워크 이름을 호스트 노드의 br-ex 브릿지에 매핑합니다. public 오버클라우드 네트워크를 물리 네트워크 datacentre에 연결하면 br-ex 브릿지를 통한 게이트웨이가 제공됩니다.

절차

1. overcloudrc 파일을 소싱합니다.

```
$ source ~/overcloudrc
```

2. provider 네트워크를 생성합니다.

- 기본 VLAN 연결에 사용할 flat 네트워크를 생성합니다.

```
(overcloud) $ openstack network create provider --external --provider-network-type flat --provider-physical-network datacentre --share
```

- 기본이 아닌 VLAN 연결에 사용할 vlan 네트워크를 생성합니다.

```
(overcloud) $ openstack network create provider --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201 --share
```

-

--provider-segment 옵션을 사용하여 사용하려는 VLAN을 정의합니다. 이 예제에서 VLAN은 201입니다.

이 예제 명령은 공유 네트워크를 생성합니다. 테넌트만 새 네트워크에 액세스할 수 있도록 --share를 지정하지 않고 테넌트를 지정할 수도 있습니다.

공급자 네트워크를 외부로 표시하는 경우 운영자만 해당 네트워크에 포트를 생성할 수 있습니다.

3. provider 네트워크에 서브넷을 추가하여 DHCP 서비스를 제공합니다.

```
(overcloud) $ openstack subnet create provider-subnet --network  provider --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
```

4. 다른 네트워크에서 공급자 네트워크를 통해 트래픽을 라우팅할 수 있도록 라우터를 생성합니다.

```
(overcloud) $ openstack router create external
```

5. 라우터의 외부 게이트웨이를 provider 네트워크로 설정합니다.

```
(overcloud) $ openstack router set --external-gateway provider external
```

6. 다른 네트워크를 이 라우터에 연결합니다. 예를 들어 subnet1 서브넷을 라우터에 연결하려면 다음 명령을 실행합니다.

```
(overcloud) $ openstack router add subnet external subnet1
```

이 명령을 수행하면 subnet1이 라우팅 테이블에 추가되고 subnet1을 사용하는 가상 머신의 트래픽이 공급자 네트워크로 라우팅됩니다.

11.6. 추가 브릿지 매핑 생성

유동 IP 네트워크는 배포 중에 추가 브릿지를 매핑한 경우 br-ex뿐만 아니라 모든 브릿지를 사용할 수 있습니다.

예를 들어 **br-floating**이라는 새 브릿지를 **floating** 물리 네트워크에 매핑하려면 환경 파일에 **NeutronBridgeMappings** 매개변수를 포함합니다.

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,floating:br-floating"
```

이 방법을 사용하면 오버클라우드를 생성한 후 별도 외부 네트워크를 생성할 수 있습니다. 예를 들어 **floating** 물리 네트워크에 매핑되는 유동 **IP** 네트워크를 생성하려면 다음 명령을 실행합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-physical-network floating --
provider-network-type vlan --provider-segment 105
(overcloud) $ openstack subnet create public --network public --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
```

11.7. 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** 사용 방법에 대한 자세한 내용은 **OpenStack Integration Test Suite Guide**를 참조하십시오.

Integration Test Suite에서 테스트에 성공하려면 몇 가지 설치 후 단계를 수행해야 합니다.

절차

1. 언더클라우드에서 이 테스트를 실행하는 경우 언더클라우드 호스트에서 오버클라우드의 내부 **API** 네트워크에 액세스할 수 있는지 확인합니다. 예를 들어 언더클라우드 호스트에 임시 **VLAN**을 추가하여 내부 **API** 네트워크(**ID**)에 액세스합니다. **201) 172.16.0.201/24** 주소 사용:

```
$ 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
```

2. **OpenStack Integration Test Suite Guide**에 설명된 대로 통합 테스트를 실행합니다.

3. 검증을 완료한 후 오버클라우드 내부 **API**에 대한 임시 연결을 삭제합니다. 이 예제에서는 다음 명령을 사용하여 이전에 생성한 **VLAN**을 언더클라우드에서 삭제합니다.

```
$ source ~/stackrc
(undercloud) $ sudo ovs-vsctl del-port vlan201
```

11.8. 오버클라우드 삭제 방지

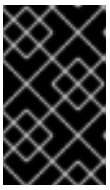
오버클라우드가 삭제되지 않도록 하려면 **heat**에 대한 사용자 지정 정책을 설정합니다.

절차

1. **prevent-stack-delete.yaml**이라는 환경 파일을 생성합니다.

2. **HeatApiPolicies** 매개변수를 설정합니다.

```
parameter_defaults:
  HeatApiPolicies:
    heat-deny-action:
      key: 'actions:action'
      value: 'rule:deny_everybody'
    heat-protect-overcloud:
      key: 'stacks:delete'
      value: 'rule:deny_everybody'
```



중요

heat-deny-action은 언더클라우드 설치에 포함해야 하는 기본 정책입니다.

3. **undercloud.conf** 파일의 **custom_env_files** 매개변수에 **prevent-stack-delete.yaml** 환경 파일을 추가합니다.

```
custom_env_files = prevent-stack-delete.yaml
```

4. 언더클라우드 설치 명령을 실행하여 구성을 새로 고칩니다.

```
$ openstack undercloud install
```

이 환경 파일을 사용하면 오버클라우드의 스택을 삭제할 수 없으므로 다음 기능을 수행할 수 없습니다.

- 오버클라우드 삭제
- 개별 **Compute** 노드 또는 **Ceph Storage** 노드 제거
- 컨트롤러 노드 교체

스택 삭제를 활성화하려면 **custom_env_files** 매개변수에서 **prevent-stack-delete.yaml** 파일을 제거하고 **openstack undercloud install** 명령을 실행합니다.

12장. 기본 오버클라우드 관리 작업 수행

이 장에서는 오버클라우드 라이프사이클 중에 수행해야 할 수 있는 기본 작업에 대해 설명합니다.

12.1. SSH를 통해 오버클라우드 노드에 액세스

SSH 프로토콜을 통해 각 오버클라우드 노드에 액세스할 수 있습니다.

- 각 오버클라우드 노드에는 **heat-admin** 사용자가 포함되어 있습니다.
- 언더클라우드의 **stack** 사용자에게 각 오버클라우드 노드의 **heat-admin** 사용자에 대한 키 기반 SSH 액세스 권한이 있습니다.
- 모든 오버클라우드 노드에는 언더클라우드가 컨트롤 플레인 네트워크에서 IP 주소로 확인되는 짧은 호스트 이름이 있습니다. 각 짧은 호스트 이름은 **.ctlplane** 접미사를 사용합니다. 예를 들어 **overcloud-controller-0**의 짧은 이름은 **overcloud-controller-0.ctlplane**입니다.

사전 요구 사항

- 작동 중인 컨트롤 플레인 네트워크가 있는 배포된 오버클라우드.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **overcloudrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 액세스할 노드의 이름을 찾습니다.

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

4. **heat-admin** 사용자로 노드에 연결하고 노드의 짧은 호스트 이름을 사용합니다.

```
(undercloud) $ ssh heat-admin@overcloud-controller-0.ctlplane
```

12.2. 컨테이너화된 서비스 관리

RHOSP(Red Hat OpenStack Platform)는 언더클라우드 및 오버클라우드 노드의 컨테이너에서 서비스를 실행합니다. 호스트의 개별 서비스를 제어해야 하는 경우도 있습니다. 이 섹션에서는 노드에서 실행하여 컨테이너화된 서비스를 관리할 수 있는 몇 가지 일반적인 명령에 대해 설명합니다.

컨테이너 및 이미지 목록 표시

실행 중인 컨테이너 목록을 표시하려면 다음 명령을 실행합니다.

```
$ sudo podman ps
```

중지되었거나 실패한 컨테이너를 명령 출력에 포함하려면 명령에 **--all** 옵션을 추가합니다.

```
$ sudo podman ps --all
```

컨테이너 이미지 목록을 표시하려면 다음 명령을 실행합니다.

```
$ sudo podman images
```

컨테이너 속성 확인

컨테이너 또는 컨테이너 이미지의 속성을 보려면 **podman inspect** 명령을 사용합니다. 예를 들어 **keystone** 컨테이너를 검사하려면 다음 명령을 실행합니다.

```
$ sudo podman inspect keystone
```

Systemd 서비스를 사용하여 컨테이너 관리

이전 버전의 **OpenStack Platform**은 **Docker** 및 해당 데몬을 사용하여 컨테이너를 관리했습니다. **OpenStack Platform 16**에서는 **Systemd** 서비스 인터페이스가 컨테이너의 라이프사이클을 관리합니다. 각 컨테이너는 서비스이며 **Systemd** 명령을 실행하여 각 컨테이너에 대해 특정 작업을 수행할 수 있습니다.

참고

Systemd는 다시 시작 정책을 적용하므로 **Podman CLI**를 사용하여 컨테이너를 중지, 시작 및 다시 시작하지 않는 것이 좋습니다. 대신 **Systemd** 서비스 명령을 사용합니다.

컨테이너 상태를 확인하려면 **systemctl status** 명령을 실행합니다.

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
  Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
  Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
Main PID: 29012 (podman)
  CGroup: /system.slice/tripleo_keystone.service
          └─29012 /usr/bin/podman start -a keystone
```

컨테이너를 중지하려면 **systemctl stop** 명령을 실행합니다.

```
$ sudo systemctl stop tripleo_keystone
```

컨테이너를 시작하려면 **systemctl start** 명령을 실행합니다.

```
$ sudo systemctl start tripleo_keystone
```

컨테이너를 다시 시작하려면 **systemctl restart** 명령을 실행합니다.

```
$ sudo systemctl restart tripleo_keystone
```

데몬이 컨테이너 상태를 모니터링하지 않으므로 **Systemd**는 다음 상황에서 대부분의 컨테이너를 자동으로 다시 시작합니다.

- **podman stop** 명령 실행과 같은 명확한 종료 코드 또는 신호
- 시작 후 **podman** 컨테이너 충돌과 같은 불명확한 종료 코드
- 불명확한 신호.

- 컨테이너를 시작하는 데 1분 30초 이상 걸리는 경우의 시간 초과

Systemd 서비스에 대한 자세한 내용은 systemd.service 문서를 참조하십시오.

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 기준으로 서비스 설정을 다시 생성하기 때문입니다. 예를 들어 keystone 컨테이너 내의 /etc/keystone/keystone.conf를 편집하고 컨테이너를 다시 시작하는 경우 컨테이너가 노드의 로컬 파일 시스템에서 /var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf를 사용하여 설정을 다시 생성합니다. 이는 다시 시작하기 전에 컨테이너 내에 만들어진 모든 변경 사항을 덮어씁니다.

Systemd 타이머를 사용하여 podman 컨테이너 모니터링

Systemd 타이머 인터페이스는 컨테이너 상태 점검을 관리합니다. 각 컨테이너에는 상태 점검 스크립트를 실행하는 서비스 장치를 실행하는 타이머가 있습니다.

모든 OpenStack Platform 컨테이너 타이머를 나열하려면 systemctl list-timers 명령을 실행하고 tripleo를 포함하는 행으로 출력을 제한합니다.

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left      Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer           tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:33 UTC  4s left      Mon 2019-02-18 20:17:03 UTC  1min 25s ago
tripleo_mistral_engine_healthcheck.timer          tripleo_mistral_engine_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left      Mon 2019-02-18 20:17:23 UTC  1min 5s ago
tripleo_keystone_healthcheck.timer            tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left      Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer           tripleo_memcached_healthcheck.service
(...)
```

특정 컨테이너 타이머의 상태를 확인하려면 healthcheck 서비스에 대해 systemctl status 명령을 실행합니다.

```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor
preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited,
status=0/SUCCESS)
```

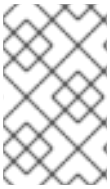
```
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
"updated": "2019-01-22T00:00:00Z", "..."}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

컨테이너 타이머를 중지, 시작 또는 다시 시작하거나 상태를 표시하려면 **.timer Systemd** 리소스에 대해 관련 **systemctl** 명령을 실행합니다. 예를 들어 **tripleo_keystone_healthcheck.timer** 리소스의 상태를 확인하려면 다음 명령을 실행합니다.

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset:
disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

상태 확인 서비스가 비활성화되었지만 해당 서비스의 타이머가 있고 활성화되어 있는 경우 현재 점검이 시간 초과되어 있어도 타이머에 따라 실행됨을 의미합니다. 검사를 수동으로 시작할 수도 있습니다.



참고

podman ps 명령은 컨테이너 상태를 표시하지 않습니다.

컨테이너 로그 확인

OpenStack Platform 16에는 모든 컨테이너의 표준 출력**(stdout)**과 각 컨테이너의 단일 파일에 통합된 표준 오류**(stderr)**가 포함된 새로운 로깅 디렉터리 **/var/log/containers/stdout**이 도입되었습니다.토

Paunch 및 **container-puppet.py** 스크립트는 출력을 **/var/log/containers/stdout** 디렉터리로 푸시하도록 **podman** 컨테이너를 구성하여 **container-puppet-*** 컨테이너와 같은 삭제된 컨테이너의 로그를 포함하여 모든 로그 컬렉션을 생성합니다.

호스트는 이 디렉터리에 로그 회전을 적용하여 용량이 큰 파일 및 디스크 공간 관련 문제를 방지합니다.

컨테이너를 교체한 경우 **podman**은 컨테이너 **ID** 대신 컨테이너 이름을 사용하므로 새 컨테이너가 동일한 로그 파일에 출력됩니다.

podman logs 명령을 사용하여 컨테이너화된 서비스의 로그를 확인할 수도 있습니다. 예를 들어

keystone 컨테이너의 로그를 보려면 다음 명령을 실행합니다.

```
$ sudo podman logs keystone
```

컨테이너 액세스

컨테이너화된 서비스 쉘에 들어가려면 **podman exec** 명령을 사용하여 **/bin/bash**를 실행합니다. 예를 들어 **keystone** 컨테이너 쉘에 들어가려면 다음 명령을 실행합니다.

```
$ sudo podman exec -it keystone /bin/bash
```

root 사용자로 **keystone** 컨테이너 쉘에 들어가려면 다음 명령을 실행합니다.

```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

컨테이너를 종료하려면 다음 명령을 실행합니다.

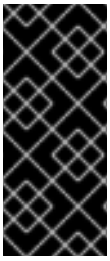
```
# exit
```

12.3. 오버클라우드 환경 수정

오버클라우드를 수정하여 추가 기능을 추가하거나 기존 작업을 변경할 수 있습니다. 오버클라우드를 수정하려면 사용자 지정 환경 파일 및 **heat** 템플릿을 수정한 다음, 초기 오버클라우드 생성 시의 **openstack overcloud deploy** 명령을 재실행합니다. 예를 들어 **7.14절. “배포 명령”**을 사용하여 오버클라우드를 생성한 경우 다음 명령을 다시 실행합니다.

```
$ 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 **heat**에서 **overcloud** 스택을 확인한 다음 스택의 각 항목을 환경 파일 및 **heat** 템플릿으로 업데이트합니다. **director**는 오버클라우드를 다시 생성하지 않고 기존 오버클라우드를 변경합니다.



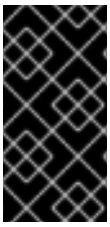
중요

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 파일을 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
```

이 명령을 수행하면 환경 파일의 새 매개변수와 리소스가 스택에 추가됩니다.



중요

director가 나중에 이러한 변경 사항을 덮어쓸 수 있으므로 오버클라우드 구성을 수동으로 변경하지 않는 것이 좋습니다.

12.4. 가상 머신을 오버클라우드로 가져오기

기존 **OpenStack** 환경에서 **RHOSP(Red Hat OpenStack Platform)** 환경으로 가상 머신을 마이그레이션할 수 있습니다.

절차

1. 기존 **OpenStack** 환경에서 실행 중인 서버의 스냅샷을 저장하여 새 이미지를 생성하고 해당 이미지를 다운로드합니다.

```
$ openstack server image create instance_name --name image_name
$ openstack image save image_name --file exported_vm.qcow2
```

2. 내보낸 이미지를 언더클라우드 노드에 복사합니다.

```
$ scp exported_vm.qcow2 stack@192.168.0.2:~/.
```

3. **stack** 사용자로 언더클라우드에 로그인합니다.

4. **overcloudrc** 파일을 소싱합니다.

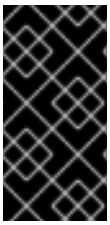
```
$ source ~/overcloudrc
```

5. 내보낸 이미지를 오버클라우드로 업로드합니다.

```
(overcloud) $ openstack image create imported_image --file exported_vm.qcow2 --disk-format qcow2 --container-format bare
```

6. 새 인스턴스를 실행합니다.

```
(overcloud) $ openstack server create  imported_instance --key-name default --flavor m1.demo --image imported_image --nic net-id=net_id
```



중요

이 명령을 수행하면 기존 **OpenStack** 환경에서 새 **Red Hat OpenStack Platform**으로 각 가상 머신 디스크가 복사됩니다. **QCOW** 스냅샷은 원래 계층화 시스템을 손실합니다.

이 프로세스에서는 컴퓨팅 노드의 모든 인스턴스를 마이그레이션합니다. 이제 인스턴스 중단 없이 노드에서 유지보수를 수행할 수 있습니다. 컴퓨팅 노드를 활성화된 상태로 되돌리려면 다음 명령을 실행합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set [hostname] nova-compute --enable
```

12.5. 동적 인벤토리 스크립트 실행

director는 **RHOSP(Red Hat OpenStack Platform)** 환경에서 **Ansible** 기반 자동화를 실행할 수 있습니다. **director**는 **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 옵션을 사용하여 모든 호스트에 대한 세부 정보를 반환합니다. 이 명령을 수행하면 동적 인벤토리가 **JSON** 포맷으로 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 사용하려는 호스트 유형으로 바꿉니다.
  - 모든 컨트롤러 노드인 경우 **controller**
  - 모든 컴퓨팅 노드인 경우 **compute**
  - 모든 오버클라우드 자식 노드인 경우 **overcloud**. 예: **controller** 및 **compute** 노드
  - 언더클라우드인 경우 **undercloud**
  - 모든 노드인 경우 **"*"**
- **[OTHER OPTIONS]**를 추가 **Ansible** 옵션으로 교체합니다.
  - **--ssh-extra-args='-o StrictHostKeyChecking=no'** 옵션을 사용하여 호스트 키 검사에서 확인을 바이패스합니다.

- **-u [USER]** 옵션을 사용하여 **Ansible** 자동화를 실행하는 **SSH** 사용자를 변경합니다. 오버클라우드의 기본 **SSH** 사용자는 동적 인벤토리에서 **ansible_ssh_user** 매개변수를 사용하여 자동으로 정의됩니다. **-u** 옵션은 이 매개변수를 덮어씁니다.

- **-m [MODULE]** 옵션을 사용하여 특정 **Ansible** 모듈을 사용합니다. 기본값은 **Linux** 명령을 실행하는 **command**입니다.

- **-a [MODULE_ARGS]** 옵션을 사용하여 선택한 모듈에 대한 인수를 정의합니다.

중요

오버클라우드의 사용자 지정 **Ansible** 자동화는 표준 오버클라우드 스택의 일부가 아닙니다. 이후 **openstack overcloud deploy** 명령을 실행하면 오버클라우드 노드에서 **OpenStack Platform** 서비스의 **Ansible** 기반 설정이 재정의될 수 있습니다.

12.6. 오버클라우드 삭제

오버클라우드를 삭제하려면 **openstack overcloud delete** 명령을 실행합니다.

절차

1. 기존 오버클라우드를 삭제합니다.

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

2. **openstack stack list** 명령 출력에 오버클라우드가 표시되지 않는 것을 확인합니다.

```
(undercloud) $ openstack stack list
```

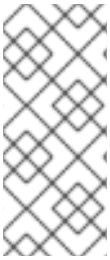
삭제에는 몇 분 정도 시간이 걸립니다.

3. 삭제가 완료되면 배포 시나리오의 표준 단계에 따라 오버클라우드를 다시 생성합니다.

13장. ANSIBLE로 오버클라우드 구성

Ansible은 오버클라우드 구성을 적용하는 기본 방식입니다. 이 장에서는 오버클라우드의 Ansible 구성을 조작하는 방법을 설명합니다.

director가 Ansible 플레이북을 자동 생성해 주지만 Ansible 구문을 숙지하는 것이 좋습니다. Ansible 사용에 관한 자세한 내용은 https://docs.ansible.com/을 참조하십시오.



참고

Ansible은 OpenStack Platform director 역할과 다른 역할의 개념을 사용합니다. Ansible 역할은 플레이북의 재사용 가능한 구성 요소를 형성하지만, director 역할은 노드 유형에 대한 OpenStack 서비스의 매핑을 포함합니다.

13.1. ANSIBLE 기반 오버클라우드 구성(CONFIG-DOWNLOAD)

config-download 기능은 director가 오버클라우드를 구성하는데 사용하는 방법입니다. director는 OpenStack Orchestration(heat) 및 OpenStack Workflow 서비스(mistral)와 함께 config-download를 사용하여 소프트웨어 구성을 생성하고 각 오버클라우드 노드에 구성을 적용합니다. heat는 SoftwareDeployment 리소스에서 모든 배포 데이터를 생성하여 오버클라우드 설치 및 설정을 수행하지만, 설정을 적용하지는 않습니다. heat는 heat API를 통해 설정 데이터만 제공합니다. director가 스택을 생성할 때 mistral 워크플로우는 heat API를 쿼리하여 설정 데이터를 가져오고 Ansible 플레이북 세트를 생성한 다음 Ansible 플레이북을 오버클라우드에 적용합니다.

결과적으로 openstack overcloud deploy 명령을 실행하면 다음 프로세스가 수행됩니다.

- director는 openstack-tripleo-heat-templates를 기반으로 새 배포 계획을 만들고, 계획을 사용자 지정할 수 있도록 모든 환경 파일과 매개변수를 추가합니다.
- director는 heat를 사용하여 배포 계획을 해석하고 오버클라우드 스택 및 모든 하위 리소스를 만듭니다. 여기에는 OpenStack Bare Metal 서비스(ironic)를 통한 프로비저닝 노드가 포함됩니다.
- heat는 배포 계획에서 소프트웨어 설정도 생성합니다. director는 이 소프트웨어 설정에서 Ansible 플레이북을 컴파일합니다.
- director는 특히 Ansible SSH 액세스를 위해 오버클라우드 노드에 임시 사용자(tripleo-admin1)를 생성합니다.

- director는 heat 소프트웨어 구성을 다운로드하고 heat 출력을 사용하여 일련의 Ansible 플레이북을 생성합니다.

- director는 ansible-playbook을 사용하여 오버클라우드 노드에 Ansible 플레이북을 적용합니다.

13.2. CONFIG-DOWNLOAD 작업 디렉터리

director는 config-download 프로세스를 위해 일련의 Ansible 플레이북을 생성합니다. 이러한 플레이북은 /var/lib/mistral/에 있는 작업 디렉터리에 저장됩니다. 이 디렉터리는 오버클라우드 이름을 따라 이름이 지정되며, 기본적으로 overcloud입니다.

작업 디렉터리에는 각 오버클라우드 역할의 이름이 붙은 여러 개의 하위 디렉터리가 존재합니다. 이러한 하위 디렉터리에는 오버클라우드 역할의 노드 설정과 관련된 모든 작업이 포함되며, 각 특정 노드의 이름이 붙은 하위 디렉터리도 추가로 포함됩니다. 해당 하위 디렉터리에는 오버클라우드 역할 작업에 적용할 노드별 변수가 포함됩니다. 그 결과 작업 디렉터리 내 오버클라우드 역할은 다음과 같이 구성됩니다.

```
─ /var/lib/mistral/overcloud
  │
  ├── Controller
  │   ├── overcloud-controller-0
  │   ├── overcloud-controller-1
  │   └── overcloud-controller-2
  ├── Compute
  │   ├── overcloud-compute-0
  │   ├── overcloud-compute-1
  │   └── overcloud-compute-2
  │
  ...
```

각각의 작업 디렉터리는 배포 작업이 끝날 때마다 변경 사항을 기록하는 로컬 Git 리포지토리입니다. 로컬 Git 리포지토리를 사용하여 각 배포 간의 설정 변경을 추적합니다.

13.3. CONFIG-DOWNLOAD 작업 디렉터리에 대한 액세스 활성화

OpenStack Workflow 서비스(mistral) 컨테이너의 mistral 사용자는 /var/lib/mistral/ 작업 디렉터리의 모든 파일을 소유합니다. 언더클라우드의 stack 사용자에게 이 디렉터리의 모든 파일에 대한 액세스 권한을 부여할 수 있습니다. 이 설정은 해당 디렉터리 내에서 특정 작업을 수행하는 데 유용합니다.

절차

1. **setfacl** 명령을 사용하여 **/var/lib/mistral** 디렉터리의 파일에 대한 액세스 권한을 언더클라우드의 **stack** 사용자에게 부여합니다.

```
$ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
$ sudo chmod -R og-rwx /var/lib/mistral/.ssh
```

이 명령을 수행하면 디렉터리에 대한 **mistral** 사용자의 액세스 권한도 유지됩니다.

13.4. CONFIG-DOWNLOAD 로그 확인

config-download 프로세스 중에 **Ansible**에서 언더클라우드의 **config-download** 작업 디렉터리에 로그 파일이 생성됩니다.

절차

1. **config-download** 작업 디렉터리 내에서 **less** 명령을 사용하여 로그를 표시합니다. 다음 예에서는 **overcloud** 작업 디렉터리를 사용하고 있습니다.

```
$ less /var/lib/mistral/overcloud/ansible.log
```

13.5. 작업 디렉터리에서 GIT 작업 수행

config-download 작업 디렉터리는 로컬 **Git** 리포지토리입니다. 배포 작업이 실행될 때마다 **director**는 관련 변경 사항이 있는 작업 디렉터리에 **Git** 커밋을 추가합니다. **Git** 작업을 수행하여 여러 단계의 배포 구성을 살펴보고, 해당 구성을 다른 배포와 비교할 수 있습니다.

작업 디렉터리에는 제한 사항이 있습니다. 예를 들어 **Git**을 사용하여 **config-download** 작업 디렉터리를 이전 버전으로 되돌리면 이 작업은 작업 디렉터리의 설정에만 영향을 미치고 다음 설정에는 영향을 미치지 않습니다.

- 오버클라우드 데이터 스키마: 이전 버전의 작업 디렉터리 소프트웨어 구성을 적용해도 데이터 마이그레이션 및 스키마 변경은 취소되지 않습니다.
- 오버클라우드의 하드웨어 레이아웃: 이전 소프트웨어 구성으로 되돌리면 오버클라우드 하드웨어와 관련된 변경(예: 확장 또는 축소)은 취소되지 않습니다.
- **heat** 스택: 작업 디렉터리의 이전 버전으로 되돌리면 **heat** 스택에 저장된 구성에는 영향을 미

치지 않습니다. **heat** 스택에서 오버클라우드에 적용되는 새로운 버전의 소프트웨어 설정을 생성합니다. 오버클라우드를 영구적으로 변경하려면 **openstack overcloud deploy** 명령을 재실행하기 전에 오버클라우드 스택에 적용한 환경 파일을 수정합니다.

config-download 작업 디렉터리의 다양한 커밋을 비교하려면 다음 단계를 완료합니다.

절차

1. 오버클라우드의 경우 **config-download** 작업 디렉터리로 변경합니다. 이 예제에서 이 작업 디렉터리는 **overcloud**라는 오버클라우드용입니다.

```
$ cd /var/lib/mistral/overcloud
```

2. **git log** 명령을 실행하여 작업 디렉터리에 커밋을 나열합니다. 로그 출력에 날짜가 표시되도록 포맷을 설정할 수도 있습니다.

```
$ git log --format=format:"%h%x09%cd%x09"
a7e9063 Mon Oct 8 21:17:52 2018 +1000
dfb9d12 Fri Oct 5 20:23:44 2018 +1000
d0a910b Wed Oct 3 19:30:16 2018 +1000
...
```

기본적으로 최근의 커밋이 가장 먼저 표시됩니다.

3. 두 커밋 해시에 대해 **git diff** 명령을 실행하여 배포 간 모든 변경 사항을 확인합니다.

```
$ git diff a7e9063 dfb9d12
```

13.6. CONFIG-DOWNLOAD를 사용하는 배포 방법

오버클라우드 배포 컨텍스트에서는 **config-download**를 사용하는 **4**가지 기본 방법이 있습니다.

표준 배포

openstack overcloud deploy 명령을 실행하여 프로비저닝 단계 이후에 구성 단계를 자동으로 실행합니다. **openstack overcloud deploy** 명령을 실행할 때 기본 방법입니다.

별도의 프로비저닝 및 구성

특정 옵션과 함께 **openstack overcloud deploy** 명령을 실행하여 프로비저닝 및 구성 단계를 구

분합니다.

배포 후 **ansible-playbook-command.sh** 스크립트 실행

통합 또는 별도의 프로비저닝 및 구성 단계를 사용하여 **openstack overcloud deploy** 명령을 실행한 다음 **config-download** 작업 디렉터리에 제공된 **ansible-playbook-command.sh** 스크립트를 실행하여 구성 단계를 다시 적용합니다.

노드를 프로비저닝하고 **config-download**를 수동으로 생성하며 **Ansible**을 실행합니다.

특정 옵션과 함께 **openstack overcloud deploy** 명령을 실행하여 노드를 프로비저닝한 다음 **deploy_steps_playbook.yaml** 플레이북을 사용하여 **ansible-playbook** 명령을 실행합니다.

13.7. 표준 배포에서 CONFIG-DOWNLOAD 실행

config-download를 실행하는 기본 방법은 **openstack overcloud deploy** 명령을 실행하는 것입니다. 이 방법은 대부분의 환경에 적합합니다.

사전 요구 사항

- 성공적인 언더클라우드 설치
- 배포할 준비가 된 오버클라우드 노드
- 특정 오버클라우드 사용자 지정과 관련된 **Heat** 환경 파일

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 배포 명령을 실행합니다. 오버클라우드에 필요한 환경 파일을 모두 포함합니다.

```
$ openstack overcloud deploy \
  --templates \
```

```
-e environment-file1.yaml \
-e environment-file2.yaml \
...
```

4. 배포 프로세스가 완료될 때까지 기다립니다.

배포 프로세스 중에 **director**는 **/var/lib/mistral/** 작업 디렉터리에 **config-download** 파일을 생성합니다. 배포 프로세스가 완료되면 작업 디렉터리의 **Ansible** 플레이북을 보고 오버클라우드를 구성하기 위해 **director**가 실행한 작업을 확인합니다.

13.8. 별도의 프로비저닝 및 구성으로 CONFIG-DOWNLOAD 실행

openstack overcloud deploy 명령에서는 **heat** 기반 프로비저닝 프로세스를 실행한 다음 **config-download** 설정 프로세스를 실행합니다. 각 프로세스를 개별적으로 실행하는 배포 명령도 실행할 수 있습니다. 오버클라우드 구성 프로세스를 실행하기 전에 노드에서 수동 사전 구성 작업을 수행할 수 있도록 이 방법으로 오버클라우드 노드를 별도의 프로세스로 프로비저닝합니다.

사전 요구 사항

- 성공적인 언더클라우드 설치
- 배포할 준비가 된 오버클라우드 노드
- 특정 오버클라우드 사용자 지정과 관련된 **Heat** 환경 파일

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **--stack-only** 옵션을 사용하여 배포 명령을 실행합니다. 오버클라우드에 필요한 환경 파일을 모두 포함합니다.

```
$ openstack overcloud deploy \
 --templates \
 -e environment-file1.yaml \
 -e environment-file2.yaml \
 ...
 --stack-only
```

4. 프로비저닝 프로세스가 완료될 때까지 대기합니다.

5. 언더클라우드에서 **tripleo-admin** 사용자의 오버클라우드로 **SSH** 액세스를 활성화합니다. **config-download** 프로세스에서는 **tripleo-admin** 사용자를 사용하여 **Ansible** 기반 설정을 수행합니다.

```
$ openstack overcloud admin authorize
```

6. 노드에서 수동 사전 구성 작업을 수행합니다. 구성에 **Ansible**을 사용하는 경우 **tripleo-admin** 사용자로 노드에 액세스합니다.

7. **--config-download-only** 옵션을 사용하여 배포 명령을 실행합니다. 오버클라우드에 필요한 환경 파일을 포함합니다.

```
$ openstack overcloud deploy \
 --templates \
 -e environment-file1.yaml \
 -e environment-file2.yaml \
 ...
 --config-download-only
```

8. 설정 프로세스가 완료될 때까지 대기합니다.

설정 단계에서 **director**는 **/var/lib/mistral/** 작업 디렉터리에 **config-download** 파일을 생성합니다. 배포 프로세스가 완료되면 작업 디렉터리의 **Ansible** 플레이북을 보고 오버클라우드를 구성하기 위해 **director**가 실행한 작업을 확인합니다.

13.9. ANSIBLE-PLAYBOOK-COMMAND.SH 스크립트를 사용하여 CONFIG-DOWNLOAD 실행

표준 방법 또는 별도의 프로비저닝 및 구성 프로세스를 사용하여 오버클라우드를 배포할 때 **director**는 **/var/lib/mistral/**에 작업 디렉터리를 생성합니다. 이 디렉터리에는 구성 프로세스를 다시 실행하는 데 필요한 플레이북과 스크립트가 포함되어 있습니다.

사전 요구 사항

- 다음 방법 중 하나로 배포된 오버클라우드:
  - 프로비저닝 및 구성 프로세스를 결합하는 표준 방법
  - 프로비저닝 및 구성 프로세스 분리

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **Ansible** 플레이북의 디렉터리로 변경합니다.

```
$ cd /var/lib/mistral/overcloud/
```

3. **/var/lib/mistral/.ssh** 디렉터리의 소유자를 **stack** 사용자로 변경합니다.

```
$ sudo chown stack. -R /var/lib/mistral/.ssh/
```

4. **ansible-playbook-command.sh** 명령을 실행하여 오버클라우드 구성을 실행합니다.

```
$ sudo ./ansible-playbook-command.sh
```

5. **/var/lib/mistral/.ssh** 디렉터리의 소유자를 **mistral** 사용자로 변경합니다. 이는 **mistral_executor** 컨테이너 내부에서 실행되는 **ansible-playbook** 명령이 성공했는지 확인하는 데 필요합니다.

```
$ sudo chown 42430:42430 -R /var/lib/mistral/.ssh/
```

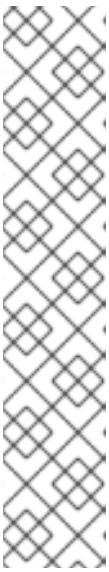
6. **mistral** 사용자로 스크립트를 다시 실행합니다.

Ansible 인수를 이 스크립트에 추가로 전달할 수 있으며, 인수는 변경되지 않은 상태로 **ansible-playbook** 명령에 전달됩니다. 즉, 확인 모드(**--check**), 호스트 제한(**--limit**), 변수 덮어쓰기(**-e**) 등의 다른 **Ansible** 기능을 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ ./ansible-playbook-command.sh --limit Controller
```

주의

--limit 를 사용하여 스케일에서 배포하는 경우 실행에 포함된 호스트만 노드의 **SSH known_hosts** 파일에 추가됩니다. 따라서 실시간 마이그레이션과 같은 일부 작업은 **known_hosts** 파일에 없는 노드에서 작동하지 않을 수 있습니다.



참고

모든 노드에서 **/etc/hosts** 파일이 최신 상태인지 확인하려면 **root** 사용자로 다음 명령을 실행합니다.

```
(undercloud)$ sudo -i
(undercloud)$ cd /var/lib/mistral/overcloud
(undercloud)$ ANSIBLE_REMOTE_USER="tripleo-admin" ansible
allovercloud \
  -i tripleo-ansible-inventory.yaml \
  -m include_role \
  -a name=tripleo-hosts-entries \
  -e @global_vars.yaml
```

7. 설정 프로세스가 완료될 때까지 대기합니다.

추가 정보

- 작업 디렉터리에는 **deploy_steps_playbook.yaml**이라는 플레이북이 포함되어 있으며 이것으로 오버클라우드 구성 작업을 관리합니다. 이 플레이북을 보려면 다음 명령을 실행합니다.

```
$ 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.

- 또한 작업 디렉터리에는 사용자 오버클라우드의 **roles_data** 파일에 정의한 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.

```
$ ls Controller/overcloud-controller-0
```

- **deploy_steps_playbook.yaml**의 **Ansible** 작업에는 태그가 지정되어 있습니다. 전체 태그 목록을 확인하려면 **ansible-playbook**과 함께 **CLI** 옵션 **--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.

```
$ ./ansible-playbook-command.sh --tags overcloud
```

1. 오버클라우드에 대해 **config-download** 플레이북을 실행하면 각 호스트의 **SSH** 지문에 대한 메시지가 표시될 수 있습니다. 이러한 메시지를 방지하려면 **ansible-playbook-command.sh** 스크립트를 실행할 때 **--ssh-common-args="-o StrictHostKeyChecking=no"**를 포함합니다.

```
$ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o
StrictHostKeyChecking=no"
```

13.10. 수동으로 생성한 플레이북을 사용하여 CONFIG-DOWNLOAD 실행

표준 워크플로우 외부에서 고유한 **config-download** 파일을 생성할 수 있습니다. 예를 들어 **--stack-only** 옵션을 사용하여 **openstack overcloud deploy** 명령을 실행하여 노드를 프로비저닝한 다음 **Ansible** 구성을 수동으로 별도로 적용할 수 있습니다.

사전 요구 사항

- 성공적인 언더클라우드 설치
- 배포할 준비가 된 오버클라우드 노드

- 특정 오버클라우드 사용자 지정과 관련된 **Heat** 환경 파일

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **--stack-only** 옵션을 사용하여 배포 명령을 실행합니다. 오버클라우드에 필요한 환경 파일을 포함합니다.

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. 프로비저닝 프로세스가 완료될 때까지 대기합니다.

5. 언더클라우드에서 **tripleo-admin** 사용자의 오버클라우드로 **SSH** 액세스를 활성화합니다. **config-download** 프로세스에서는 **tripleo-admin** 사용자를 사용하여 **Ansible** 기반 설정을 수행합니다.

```
$ openstack overcloud admin authorize
```

6. **config-download** 파일을 생성합니다.

```
$ openstack overcloud config download \
  --name overcloud \
  --config-dir ~/config-download
```

- **--name**은 **Ansible** 파일 내보내기에 사용할 오버클라우드의 이름입니다.
- **--config-dir**은 **config-download** 파일을 저장할 위치입니다.

7. **config-download** 파일이 포함된 디렉터리로 변경합니다.

```
$ cd ~/config-download
```

8. 정적 인벤토리 파일을 생성합니다.

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user heat-admin \
  --static-yaml-inventory inventory.yaml
```

- **<overcloud>**를 해당 오버클라우드 이름으로 교체합니다.

9. **config-download** 파일과 정적 인벤토리 파일을 사용하여 설정을 수행합니다. 배포용 플레이북을 실행하려면 **ansible-playbook** 명령을 실행합니다.

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/config-download/deploy_steps_playbook.yaml
```

10. 설정 프로세스가 완료될 때까지 대기합니다.

11. 이 설정에서 **overcloudrc** 파일을 수동으로 생성하려면 다음 명령을 실행합니다.

```
$ openstack action execution run \
  --save-result \
  --run-sync \
  tripleo.deployment.overcloudrc \
  '{"container":"overcloud"}' \
  | jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. 배포 상태를 **success**로 수동으로 설정합니다.

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<OVERCLOUD>"}'
```

- **<OVERCLOUD>**를 해당 오버클라우드 이름으로 교체합니다.

추가 정보

- **config-download** 디렉터리에는 **deploy_steps_playbook.yaml**이라는 플레이북이 포함되어 있으며 이는 오버클라우드 구성을 실행합니다. 이 플레이북을 보려면 다음 명령을 실행합니다.

```
$ 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.

- 또한 **config-download** 디렉터리에는 사용자 오버클라우드의 **roles_data** 파일에 정의한 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 포멧을 사용합니다.

```
$ ls Controller/overcloud-controller-0
```

- **deploy_steps_playbook.yaml**의 **Ansible** 작업에는 태그가 지정되어 있습니다. 전체 태그 목록을 확인하려면 **ansible-playbook**과 함께 **CLI** 옵션 **--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.

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

1. 오버클라우드에 대해 **config-download** 플레이북을 실행하면 각 호스트의 **SSH** 지문에 대한 메시지가 표시될 수 있습니다. 이러한 메시지가 표시되지 않게 하려면 **ansible-playbook** 명령에 **--ssh-common-args="-o StrictHostKeyChecking=no"**를 포함합니다.

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

13.11. CONFIG-DOWNLOAD의 제한 사항

config-download 기능에는 몇 가지 제한 사항이 있습니다.

- **--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.

- 작업 이름에 변수를 사용하는 특정 작업에 **--start-at-task** 인수를 사용할 수 없습니다. 예를 들어 **--start-at-task** 인수는 다음 **Ansible** 작업에는 작동하지 않습니다.

```
- name: Run puppet host configuration for step {{ step }}
```

- 오버클라우드 배포에 **director**가 배포한 **Ceph Storage** 클러스터가 포함된 경우 **external_deploy_steps** 작업을 건너뛰지 않으면 **--check** 옵션을 사용할 때 **step1** 작업을 건너뛸 수 없습니다.

- **--forks** 옵션을 사용하여 병렬 **Ansible** 작업 수를 설정할 수 있습니다. 그러나 **config-download** 작업의 성능은 **25**개의 병렬 작업 후에 성능이 저하됩니다. 이러한 이유로 **--forks** 옵션을 사용하여 **25**개의 작업 수를 초과하지 마십시오.

13.12. CONFIG-DOWNLOAD 최상위 파일

다음 파일은 **config-download** 작업 디렉터리에 있는 중요한 최상위 파일입니다.

Ansible 구성 및 실행

다음 파일은 **config-download** 작업 디렉터리에 있으며, **Ansible**을 구성하고 실행하기 위한 파일입니다.

ansible.cfg

ansible-playbook을 실행할 때 사용되는 구성 파일입니다.

ansible.log

ansible-playbook 마지막 실행 시 생성된 로그 파일입니다.

ansible-errors.json

모든 배포 오류가 포함된 **JSON** 구조 파일입니다.

ansible-playbook-command.sh

마지막 배포 작업의 **ansible-playbook** 명령을 재실행하기 위한 실행 스크립트입니다.

ssh_private_key

Ansible에서 오버클라우드 노드에 액세스하는 데 사용하는 개인 **SSH** 키입니다.

tripleo-ansible-inventory.yaml

모든 오버클라우드 노드에 대한 호스트 및 변수가 포함된 **Ansible** 인벤토리 파일입니다.

overcloud-config.tar.gz

작업 디렉터리의 아카이브입니다.

플레이북

다음 파일은 **config-download** 작업 디렉터리에 있는 플레이북입니다.

deploy_steps_playbook.yaml

주요 배포 단계입니다. 이 플레이북은 오버클라우드에 대한 주요 구성 작업을 수행합니다.

pre_upgrade_rolling_steps_playbook.yaml

메이저 업그레이드를 위한 사전 업그레이드 단계입니다.

upgrade_steps_playbook.yaml

메이저 업그레이드 단계입니다.

post_upgrade_steps_playbook.yaml

메이저 업그레이드 후 업그레이드 단계입니다.

update_steps_playbook.yaml

마이너 업데이트 단계입니다.

fast_forward_upgrade_playbook.yaml

Fast Forward Upgrade 작업입니다. 이 플레이북은 장기 버전의 **Red Hat OpenStack Platform** 을 다음 버전으로 업그레이드하려는 경우에만 사용합니다.

13.13. CONFIG-DOWNLOAD 태그

플레이북은 태그된 작업을 사용하여 오버클라우드에 적용되는 작업을 제어합니다. **ansible-playbook CLI** 인수 **--tags** 또는 **--skip-tags** 태그를 사용하여 실행할 작업을 제어합니다. 다음 목록에는 기본적으로 활성화된 태그에 대한 정보가 포함되어 있습니다.

facts

사실 수집 작업입니다.

common_roles

모든 노드에 공통된 **Ansible** 역할입니다.

overcloud

오버클라우드 배포에 대한 모든 플레이입니다.

pre_deploy_steps

deploy_steps 작업 이전에 발생한 배포입니다.

host_prep_steps

호스트 준비 단계입니다.

deploy_steps

배포 단계입니다.

post_deploy_steps

deploy_steps 작업 이후에 수행되는 단계입니다.

external

모든 외부 배포 작업입니다.

external_deploy_steps

언더클라우드에서만 실행되는 외부 배포 작업입니다.

13.14. CONFIG-DOWNLOAD 배포 단계

deploy_steps_playbook.yaml 플레이북은 오버클라우드를 구성합니다. 이 플레이북은 오버클라우드 배포 계획에 따라 전체 오버클라우드를 배포하는 데 필요한 모든 소프트웨어 설정에 적용됩니다.

이 섹션에는 이 플레이북 내에서 사용되는 다양한 **Ansible** 플레이가 요약되어 있습니다. 이 섹션에 있는 플레이 이름은 플레이북에서 사용되고 **ansible-playbook** 출력에 표시되는 이름과 같습니다. 이 섹션에는 각 플레이에 설정된 **Ansible** 태그에 대한 정보도 포함되어 있습니다.

Gather facts from undercloud

언더클라우드 노드에 대한 정보 수집입니다.

태그: **facts**

Gather facts from overcloud

오버클라우드 노드에 대한 정보 수집입니다.

태그: **facts**

Load global variables

global_vars.yaml에서 모든 변수를 로드합니다.

태그: **always**

Common roles for TripleO servers

모든 오버클라우드 노드에 일반적인 **Ansible** 역할을 적용합니다. **Bootstrap** 패키지 설치를 위한 **tripleo-bootstrap**과 알려진 **SSH** 호스트 구성을 위한 **tripleo-ssh-known-hosts**가 포함됩니다.

태그: **common_roles**

Overcloud deploy step tasks for step 0

deploy_steps_tasks 템플릿 인터페이스의 작업을 적용합니다.

태그: **overcloud, deploy_steps**

Server deployments

네트워킹 및 **hieradata**와 같은 구성에 서버별 **heat** 배포를 적용합니다. **NetworkDeployment, <Role>Deployment, <Role>AllNodesDeployment** 등을 포함합니다.

태그: **overcloud, pre_deploy_steps**

Host prep steps

host_prep_steps 템플릿 인터페이스의 작업을 적용합니다.

태그: **overcloud, host_prep_steps**

External deployment step [1,2,3,4,5]

external_deploy_steps_tasks 템플릿 인터페이스에서 작업을 적용합니다. **Ansible**은 언더클라우드 노드에서만 이 작업을 실행합니다.

태그: **external, external_deploy_steps**

Overcloud deploy step tasks for [1,2,3,4,5]

deploy_steps_tasks 템플릿 인터페이스의 작업을 적용합니다.

태그: **overcloud, deploy_steps**

Overcloud common deploy step tasks [1,2,3,4,5]

각 단계에서 수행되는 공통 작업을 적용합니다. **puppet** 호스트 설정, **container-puppet.py**, **paunch**(컨테이너 설정) 등이 포함됩니다.

태그: **overcloud, deploy_steps**

Server Post Deployments

5단계 배포 프로세스 이후 수행되는 구성에 대해 서버별 **heat** 배포를 적용합니다.

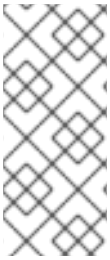
태그: **overcloud, post_deploy_steps**

External deployment Post Deploy tasks

external_post_deploy_steps_tasks 템플릿 인터페이스에서 작업을 적용합니다. **Ansible**은 언더클라우드 노드에서만 이 작업을 실행합니다.

태그: **external, external_deploy_steps**

14장. ANSIBLE을 사용하여 컨테이너 관리



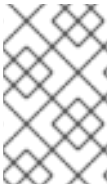
참고

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

Red Hat OpenStack Platform 16.2는 **Paunch**를 사용하여 컨테이너를 관리합니다. 그러나 **Ansible** 역할 **tripleo-container-manage**를 사용하여 컨테이너에서 관리 작업을 수행할 수도 있습니다. **tripleo-container-manage** 역할을 사용하려면 먼저 **Paunch**를 비활성화해야 합니다. **Paunch**가 비활성화되면 **director**는 **Ansible** 역할을 자동으로 사용하며 특정 컨테이너 관리 작업을 수행하기 위해 사용자 지정 플레이북을 작성할 수도 있습니다.

- **heat**가 생성하는 컨테이너 설정 데이터를 수집합니다. **tripleo-container-manage** 역할은 이 데이터를 사용하여 컨테이너 배포를 오케스트레이션합니다.
- 컨테이너를 시작합니다.
- 컨테이너를 중지합니다.
- 컨테이너를 업데이트합니다.
- 컨테이너를 삭제합니다.
- 특정 구성으로 컨테이너를 실행합니다.

director가 컨테이너 관리를 자동으로 수행하지만 컨테이너 구성을 사용자 지정하거나 오버클라우드를 재배포하지 않고 컨테이너에 핫픽스를 적용할 수 있습니다.



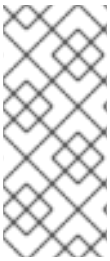
참고

이 역할은 **Podman** 컨테이너 관리만 지원합니다.

사전 요구 사항

- 성공적인 언더클라우드 설치 자세한 내용은 4.8절. “director 설치” 의 내용을 참조하십시오.

14.1. 언더클라우드에서 TRIPLEO-CONTAINER-MANAGE ANSIBLE 역할 활성화



참고

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

Paunch는 **Red Hat OpenStack Platform 16.2**의 기본 컨테이너 관리 메커니즘입니다. 그러나 **tripleo-container-manage Ansible** 역할을 사용할 수도 있습니다. 이 역할을 사용하려면 **Paunch**를 비활성화해야 합니다.

사전 요구 사항

- 기본 운영 체제 및 **python3-tripleoclient** 패키지가 설치된 호스트 머신. 자세한 내용은 3장. *director 설치 준비*의 내용을 참조하십시오.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **undercloud.conf** 파일에서 **undercloud_enable_paunch** 매개변수를 **false**로 설정합니다.

```
undercloud_enable_paunch: false
```

3. **openstack undercloud install** 명령을 실행합니다.

```
$ openstack undercloud install
```

14.2. 오버클라우드에서 TRIPLEO-CONTAINER-MANAGE ANSIBLE 역할 활성화

참고

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

Paunch는 **Red Hat OpenStack Platform 16.2**의 기본 컨테이너 관리 메커니즘입니다. 그러나 **tripleo-container-manage Ansible** 역할을 사용할 수도 있습니다. 이 역할을 사용하려면 **Paunch**를 비활성화해야 합니다.

사전 요구 사항

- 성공적인 언더클라우드 설치 자세한 내용은 **4장. *언더클라우드에 director 설치*** 의 내용을 참조하십시오.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **stackrc** 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 해당 배포와 관련된 다른 모든 환경 파일과 함께 **/usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml** 파일을 오버클라우드 배포 명령에 추가합니다.

```
(undercloud) [stack@director ~]$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml
  -e <other_environment_files>
  ...
```

14.3. 단일 컨테이너에서 작업 수행

참고

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

tripleo-container-manage 역할을 사용하여 모든 컨테이너 또는 특정 컨테이너를 관리할 수 있습니다. 특정 컨테이너를 관리하려면 사용자 지정 **Ansible** 플레이북으로 특정 컨테이너를 대상으로 지정할 수 있도록 컨테이너 배포 단계 및 컨테이너 구성 **JSON** 파일의 이름을 확인해야 합니다.

사전 요구 사항

- 성공적인 언더클라우드 설치 자세한 내용은 *4장. 언더클라우드에 director 설치* 의 내용을 참조하십시오.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **overcloudrc** 인증 정보 파일을 소싱합니다.

```
$ source ~/overcloudrc
```

3. 컨테이너 배포 단계를 확인합니다. **/var/lib/tripleo-config/container-startup-config/step_{1,2,3,4,5,6}** 디렉터리에서 각 단계의 컨테이너 구성을 찾을 수 있습니다.

4. 컨테이너의 **JSON** 구성 파일을 확인합니다. 관련 **step_*** 디렉터리에서 컨테이너 구성 파일을 찾을 수 있습니다. 예를 들어 **1**단계에서 **HAProxy** 컨테이너의 구성 파일은 **/var/lib/tripleo-config/container-startup-config/step_1/haproxy.json** 입니다.

5. 적절한 **Ansible** 플레이북을 작성합니다. 예를 들어 **HAProxy** 컨테이너 이미지를 바꾸려면 다음 샘플 플레이북을 사용합니다.

```
- hosts: localhost
  become: true
  tasks:
    - name: Manage step_1 containers using tripleo-ansible
      block:
        - name: "Manage HAproxy container at step 1 with tripleo-ansible"
          include_role:
            name: tripleo-container-manage
          vars:
            tripleo_container_manage_systemd_order: true
            tripleo_container_manage_config_patterns: 'haproxy.json'
            tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-
config/step_1"
            tripleo_container_manage_config_id: "tripleo_step1"
```

```
tripleo_container_manage_config_overrides:
  haproxy:
    image: registry.redhat.io/tripleomaster/<HAProxy-container>:hotfix
```

tripleo-container-manage 역할과 함께 사용할 수 있는 변수에 대한 자세한 내용은 **14.4절. "tripleo-container-manage 역할 변수"**을 참조하십시오.

6. 플레이북을 실행합니다.

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml
```

변경 사항을 적용하지 않고 플레이북을 실행하려면 **ansible-playbook** 명령에 **--check** 옵션을 추가합니다.

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check
```

변경 사항을 적용하지 않고 플레이북이 컨테이너에 적용하는 변경 사항을 파악하려면 **ansible-playbook** 명령에 **--check** 및 **--diff** 옵션을 추가합니다.

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check --diff
```

14.4. TRIPLEO-CONTAINER-MANAGE 역할 변수

참고

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

tripleo-container-manage Ansible 역할에는 다음 변수가 포함됩니다.

표 14.1. 역할 변수

이름	기본값	설명

이름	기본값	설명
tripleo_container_manage_check_puppet_config	false	Ansible에서 Puppet 컨테이너 구성을 확인하도록 하려면 이 변수를 사용합니다. Ansible은 구성 해시를 사용하여 업데이트된 컨테이너 구성을 식별할 수 있습니다. 컨테이너에 Puppet의 새로운 구성이 있는 경우 Ansible이 새 구성을 감지하고 Ansible이 다시 시작해야 하는 컨테이너 목록에 컨테이너를 추가할 수 있도록 이 변수를 **true**로 설정합니다.
tripleo_container_manage_cli	podman	컨테이너를 관리하는 데 사용할 명령행 인터페이스를 설정하려면 이 변수를 사용합니다. **tripleo-container-manage** 역할은 Podman만 지원합니다.
tripleo_container_manage_concurrency	1	동시에 관리하려는 컨테이너 수를 설정하려면 이 변수를 사용합니다.
tripleo_container_manage_config	/var/lib/tripleo-config/	컨테이너 구성 디렉터리의 경로를 설정하려면 이 변수를 사용합니다.
tripleo_container_manage_config_id	tripleo	특정 구성 단계의 ID를 설정하려면 이 변수를 사용합니다. 예를 들어 배포 2단계의 컨테이너를 관리하려면 이 값을 **tripleo_step2**로 설정합니다.
tripleo_container_manage_config_patterns	*.json	컨테이너 구성 디렉터리에서 구성 파일을 식별하는 bash 정규식을 설정하려면 이 변수를 사용합니다.
tripleo_container_manage_debug	false	디버그 모드를 활성화 또는 비활성화하려면 이 변수를 사용합니다. 특정 일회성 구성을 사용하여 컨테이너를 실행하거나, 컨테이너의 수명 주기를 관리하는 컨테이너 명령을 출력하거나, 테스트 및 검증 목적으로 no-op 컨테이너 관리 작업을 실행하려면 디버그 모드에서 **tripleo-container-manage** 역할을 실행합니다.
tripleo_container_manage_healthcheck_disable	false	이 변수를 사용하여 상태 확인을 활성화하거나 비활성화합니다.

이름	기본값	설명
tripleo_container_manage_log_path	/var/log/containers/stdouts	이 변수를 사용하여 컨테이너의 stdout 로그 경로를 설정합니다.
tripleo_container_manage_systemd_order	false	이 변수를 사용하여 Ansible에서 systemd 종료 순서를 활성화하거나 비활성화합니다.
tripleo_container_manage_systemd_teardown	true	이 변수를 사용하여 더 이상 사용되지 않는 컨테이너를 트리거합니다.
tripleo_container_manage_config_overrides	{}	이 변수를 사용하여 컨테이너 구성을 덮어쓰기합니다. 이 변수는 각 키가 컨테이너 이름인 값과 컨테이너 이미지 또는 사용자와 같이 덮어쓰려는 매개변수의 사전을 가져옵니다. 이 변수는 JSON 컨테이너 구성 파일에 사용자 지정 덮어쓰기를 쓰지 않으며 새로운 컨테이너 배포, 업데이트 또는 업그레이드는 JSON 구성 파일의 콘텐츠로 되돌아갑니다.
tripleo_container_manage_valid_exit_code	[]	이 변수를 사용하여 컨테이너가 종료 코드를 반환하는지 확인합니다. 이 값은 목록이어야 합니다(예: **[0,3]**).

15장. 검증 프레임워크 사용

Red Hat OpenStack Platform에는 언더클라우드 및 오버클라우드의 요구 사항과 기능을 확인하는 데 사용할 수 있는 검증 프레임워크가 포함되어 있습니다. 프레임워크에는 다음과 같은 두 가지 검증 유형이 포함됩니다.

- 수동 Ansible 기반 검증은 openstack tripleo validator 명령 세트를 통해 실행합니다.
- 자동 진행 중 검증은 배포 프로세스 중 실행합니다.

실행할 검증을 이해하고 해당 환경과 관련이 없는 검증을 건너뛰어야 합니다. 예를 들어 사전 배포 검증에는 TLS-everywhere에 대한 테스트가 포함됩니다. TLS-everywhere에 대한 환경을 구성하지 않으려면 이 테스트가 실패합니다. openstack tripleo validator run 명령의 --validation 옵션을 사용하여 환경에 따라 유효성 검사를 구체화합니다.

15.1. ANSIBLE 기반 검증

Red Hat OpenStack Platform director 설치 중에 director는 openstack-tripleo-validations 패키지에서 Playbook 세트도 설치합니다. 각 Playbook에는 특정 시스템 요구 사항의 테스트와 테스트 실행 시기를 정의하는 그룹 세트가 포함되어 있습니다.

no-op

no-op(작업 없음) 작업을 실행하여 워크플로우가 제대로 작동하는지 확인하는 검증입니다. 이 검증은 언더클라우드 및 오버클라우드 모두에서 실행됩니다.

prep

언더클라우드 노드의 하드웨어 구성을 확인하는 검증입니다. openstack undercloud install 명령을 실행하기 전에 이 검증을 실행합니다.

openshift-on-openstack

OpenStack에 OpenShift를 배포할 수 있도록 환경이 요구 사항을 충족하는지 확인하는 검증입니다.

pre-introspection

Ironic Inspector를 사용하여 노드 인트로스펙션 전에 실행할 검증입니다.

pre-deployment

openstack overcloud deploy 명령 전에 실행할 검증입니다.

post-deployment

오버클라우드 배포를 완료한 후 실행할 검증입니다.

pre-upgrade

업그레이드 전에 **OpenStack** 배포를 확인하는 검증입니다.

post-upgrade

업그레이드 후에 **OpenStack** 배포를 확인하는 검증입니다.

15.2. 검증 나열

openstack tripleo validator list 명령을 실행하여 사용 가능한 여러 다른 유형의 검증을 나열합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

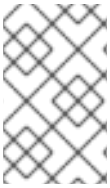
2. **openstack tripleo validator list** 명령을 실행합니다.

- 모든 검증을 나열하려면 옵션 없이 명령을 실행합니다.

```
$ openstack tripleo validator list
```

- 그룹의 검증을 나열하려면 **--group** 옵션을 사용하여 명령을 실행합니다.

```
$ openstack tripleo validator list --group prep
```



참고

전체 옵션 목록은 **openstack tripleo validator list --help**를 실행하십시오.

15.3. 검증 실행

검증 또는 검증 그룹을 실행하려면 **openstack tripleo validator run** 명령을 사용합니다. 전체 옵션 목록을 보려면 **openstack tripleo validator run --help** 명령을 사용합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. **inventory.yaml** 이라는 정적 인벤토리 파일을 생성하고 검증합니다.

```
$ tripleo-ansible-inventory --static-yaml-inventory inventory.yaml
$ openstack tripleo validator run  --group pre-introspection -i inventory.yaml
```

3. **openstack tripleo validator run** 명령을 입력합니다.

- 단일 검증을 실행하려면 **--validation-name** 옵션과 검증 이름을 사용하여 명령을 실행합니다. 예를 들어 각 노드의 메모리 요구 사항을 확인하려면 **--validation check-ram**을 입력합니다.

```
$ openstack tripleo validator run --validation check-ram
```

오버클라우드에서 기본 계획 이름 **overcloud** 이외의 다른 계획 이름을 사용하려면 다음과 같이 **--plan** 옵션을 사용하여 계획 이름을 설정합니다.

```
$ openstack tripleo validator run --validation check-ram --plan myovercloud
```

여러 특정 검증을 실행하려면 실행하려는 쉼표로 구분된 검증 목록과 함께 **--validation** 옵션을 사용합니다. 사용 가능한 검증 목록을 확인하는 방법에 대한 자세한 내용은 검증 목록을 참조하십시오.

- 그룹의 검증을 모두 실행하려면 **--group** 옵션을 사용하여 명령을 입력합니다.

```
$ openstack tripleo validator run --group prep
```

특정 검증의 자세한 출력을 보려면 보고서의 특정 검증 **UUID**에 대해 **openstack tripleo validator show run --full** 명령을 실행합니다.

```
$ openstack tripleo validator show run --full <UUID>
```

15.4. 검증 내역 보기

검증 또는 검증 그룹을 실행한 후 **director**는 각 검증 결과를 저장합니다. **openstack tripleo validator show history** 명령을 사용하여 지난 검증 결과를 봅니다.

사전 요구 사항

- 검증 또는 검증 그룹을 실행해야 합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 모든 검증 목록을 확인합니다.

```
$ openstack tripleo validator show history
```

특정 검증 유형의 기록을 보려면 **--validation** 옵션을 사용하여 동일한 명령을 실행합니다.

```
$ openstack tripleo validator show history --validation ntp
```

3. **openstack tripleo validator show run --full** 명령을 사용하여 특정 검증 **UUID**의 로그를 확인합니다.

```
$ openstack tripleo validator show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
```

15.5. 검증 프레임워크 로그 포멧

검증 또는 검증 그룹을 실행한 후 **director**는 **/var/logs/validations** 디렉터리의 각 검증에서 **JSON** 형식의 로그를 저장합니다. 파일을 수동으로 확인하거나 **openstack tripleo validator show run --full** 명령

을 사용하여 특정 검증 UUID에 대한 로그를 표시할 수 있습니다.

각 검증 로그 파일은 특정 포맷을 따릅니다.

- <UUID>_<Name>_<Time>

UUID

검증을 위한 Ansible UUID입니다.

이름

검증을 위한 Ansible 이름입니다.

시간

검증을 실행한 경우의 시작 날짜 및 시간입니다.

각 검증 로그에는 다음 세 가지 주요 부분이 포함됩니다.

- 플레이
- 통계
- validation_output

플레이

plays 섹션에는 director가 검증의 일부로 수행한 작업에 대한 정보가 포함되어 있습니다.

play

플레이는 작업 그룹입니다. 각 play 섹션에는 시작 및 종료 시간, 기간, 플레이의 호스트 그룹, 검증 ID 및 경로를 비롯한 특정 작업 그룹에 대한 정보가 포함되어 있습니다.

tasks

검증을 수행하기 위해 director가 실행하는 개별 Ansible 작업입니다. 각 tasks 섹션에는 각 개별

호스트에서 발생한 작업과 작업 실행 결과가 포함된 **hosts** 섹션이 포함되어 있습니다. **tasks** 섹션에는 작업 기간이 포함된 **task** 섹션도 포함되어 있습니다.

통계

stats 섹션에는 각 호스트에 있는 모든 작업의 결과(예: 성공 및 실패한 작업)에 대한 기본 요약이 포함되어 있습니다.

validation_output

검증 중에 작업이 실패하거나 경고 메시지가 발생한 경우 **validation_output**에는 해당 실패 또는 경고의 출력이 포함됩니다.

15.6. 검증 프레임워크 로그 출력 포맷

유효성 검사 프레임워크의 기본 동작은 유효성 검사 로그를 **JSON** 형식으로 저장하는 것입니다. **ANSIBLE_STDOUT_CALLBACK** 환경 변수를 사용하여 로그 출력을 변경할 수 있습니다.

검증 출력 로그 포맷을 변경하려면 검증을 실행하고 **--extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback>** 옵션을 포함합니다.

```
$ openstack tripleo validator run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback> --validation check-ram
```

- **<callback>**을 **Ansible** 출력 콜백으로 바꿉니다. 표준 **Ansible** 출력 콜백 목록을 보려면 다음 명령을 실행합니다.

```
$ ansible-doc -t callback -l
```

검증 프레임워크에는 다음과 같은 추가 콜백이 포함됩니다.

validation_json

프레임워크는 **JSON** 형식의 검증 결과를 **/var/logs/validations**의 로그 파일로 저장합니다. 이는 검증 프레임워크의 기본 콜백입니다.

validation_stdout

프레임워크는 화면에 **JSON** 형식의 검증 결과를 표시합니다.

http_json

프레임워크는 **JSON** 형식의 검증 결과를 외부 로깅 서버로 전송합니다. 이 콜백에 대한 추가 환경 변수도 포함해야 합니다.

HTTP_JSON_SERVER

외부 서버의 **URL**입니다.

HTTP_JSON_PORT

외부 서버의 **API** 진입점에 대한 포트입니다. **8989**의 기본 포트입니다.

추가 **--extra-env-vars** 옵션을 사용하여 이러한 환경 변수를 설정합니다.

```
$ openstack tripleo validator run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=http_json \
  --extra-env-vars HTTP_JSON_SERVER=http://logserver.example.com \
  --extra-env-vars HTTP_JSON_PORT=8989 \
  --validation check-ram
```

중요

http_json 콜백을 사용하기 전에 **ansible.cfg** 파일의 **callback_whitelist** 매개변수에 **http_json**을 추가해야 합니다.

```
callback_whitelist = http_json
```

15.7. 진행 중인 검증

Red Hat OpenStack Platform에는 구성 가능 서비스 템플릿에 진행 중 검증이 포함되어 있습니다. 진행 중 검증은 오버클라우드 배포 프로세스의 주요 단계에서 서비스의 작동 상태를 확인합니다.

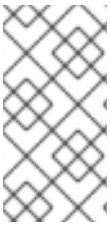
진행 중 검증은 배포 프로세스의 일부로 자동 실행됩니다. 일부 진행 중 검증에서는 **openstack-tripleo-validations** 패키지의 역할도 사용합니다.

16장. 오버클라우드 노드 확장

오버클라우드 생성 후 노드를 추가하거나 삭제하려면 오버클라우드를 업데이트해야 합니다.

주의

오버클라우드에서 노드를 삭제하는 데 **openstack server delete**를 사용하지 마십시오. 이 섹션의 절차에 따라 노드를 올바르게 삭제 및 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 16.1. 각 노드 유형의 확장 지원

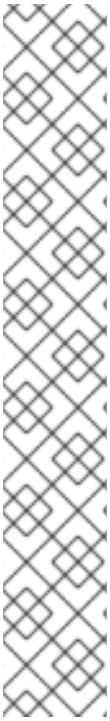
노드 유형	확장 가능 여부	축소 가능 여부	참고
컨트롤러	N	N	17장. *컨트롤러 노드 교체*의 절차를 사용하여 컨트롤러 노드를 교체할 수 있습니다.
Compute	Y	Y	
Ceph Storage 노드	Y	N	초기 오버클라우드 생성 시 적어도 하나의 Ceph Storage 노드가 있어야 합니다.
Object Storage 노드	Y	Y	

중요

오버클라우드를 확장하기 전에 **10GB** 이상의 사용 가능한 공간이 있어야 합니다. 이 공간은 노드 프로비저닝 프로세스 중에 이미지 변환 및 캐싱에 사용됩니다.

16.1. 오버클라우드에 노드 추가

director 노드 풀에 노드를 추가하려면 다음 단계를 완료합니다.



참고

Red Hat OpenStack Platform의 새로운 설치에는 보안 에라타 및 버그 수정과 같은 특정 업데이트가 포함되어 있지 않습니다. 결과적으로 **Red Hat** 고객 포털 또는 **Red Hat Satellite Server**를 사용하는 연결된 환경을 확장하는 경우 **RPM** 업데이트가 새 노드에 적용되지 않습니다. 최신 업데이트를 오버클라우드 노드에 적용하려면 다음 중 하나를 수행해야 합니다.

- 스케일 아웃 작업 후 노드의 오버클라우드 업데이트를 완료합니다.
- **virt-customize** 툴을 사용하여 패키지를 스케일 아웃 작업 전에 기본 오버클라우드 이미지로 수정합니다. 자세한 내용은 **Red Hat Knowledgebase** 솔루션 **virt-customize**를 사용하여 **Red Hat Linux OpenStack Platform Overcloud** 이미지 수정 을 참조하십시오.

절차

1. 등록하려는 새 노드의 세부 정보가 포함된 **newnodes.json** 이라는 새 **JSON** 파일을 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. 새 노드를 등록합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud node import newnodes.json
```

3. 새 노드를 등록한 후 각 새 노드에 인트로스펙션 프로세스를 시작합니다.

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

이 프로세스에서 노드의 하드웨어 속성을 감지하여 벤치마킹합니다.

4. 노드의 이미지 속성을 설정합니다.

```
(undercloud) $ openstack overcloud node configure [NODE UUID]
```

16.2. 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 수가 포함된 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]
```

오버클라우드를 처음 생성할 때의 모든 환경 파일과 옵션을 포함해야 합니다. 여기에는 컴퓨팅 이외의 노드에 대한 동일한 확장 매개변수가 포함됩니다.

4. 배포 작업이 완료될 때까지 기다립니다.

16.3. 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 nodes**를 참조하십시오.

- **Instance HA**가 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 **UUID**를 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 가장 최근 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 |...]
```

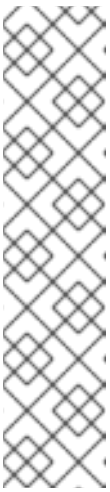
참고

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 **UUID**로 바꿉니다.
- **& lt;node_1 >**을 바꾸고 선택적으로 **[node_n]** 까지 모든 노드를 삭제하려는 컴퓨팅 노드의 **Compute** 서비스 호스트 이름 또는 **UUID**로 바꿉니다. **UUID**와 호스트 이름을 혼합하여 사용하지 마십시오. **UUID**만 사용하거나 호스트 이름만 사용하십시오.



참고

노드의 전원이 이미 꺼진 경우 이 명령은 경고 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. **삭제한 각 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
```

8. **노드 삭제가 완료되면 오버클라우드 스택의 상태를 확인합니다.**

```
(undercloud)$ openstack stack list
```

표 16.2. 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 오버클라우드에서 수동으로 컴퓨팅 노드 제거를 참조하십시오.

9. **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>
```

10. 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 **4**개의 노드에서 **3**개의 노드로 줄입니다.

```
parameter_defaults:
  ...
  ComputeCount: 3
```

노드 개수를 줄이면 **openstack overcloud deploy**를 실행할 때 **director**에서 새 노드를 프로비저닝하지 않습니다.

오버클라우드 배포에서 삭제된 컴퓨팅 노드를 교체하는 경우 삭제된 컴퓨팅 노드 배치를 참조하십시오.

16.3.1. 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 **UUID**를 확인합니다.

```
(undercloud)$ openstack stack list
```

2. 수동으로 삭제하려는 노드의 **UUID**를 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 **4**분이 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. 노드에서 **Compute** 서비스를 삭제합니다.

```
(overcloud)$ openstack compute service delete <service_id>
```

9. 배치 서비스에서 리소스 공급자로 삭제된 **Compute** 서비스를 제거합니다.

```
(overcloud)$ openstack resource provider list
(overcloud)$ openstack resource provider delete <uuid>
```

10. **source** 명령으로 언더클라우드 설정을 로드합니다.

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

11. 스택에서 컴퓨팅 노드를 삭제합니다.

```
(undercloud)$ openstack overcloud node delete --stack <overcloud> <node>
```

- **<overcloud>**를 오버클라우드 스택의 이름 또는 **UUID**로 바꿉니다.
- **& lt;node >**를 삭제하려는 컴퓨팅 노드의 **Compute** 서비스 호스트 이름 또는 **UUID**로 바꿉니다.

참고

노드의 전원이 이미 꺼진 경우 이 명령은 경고 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.

12. 오버클라우드 노드가 삭제될 때까지 기다립니다.

13. 노드 삭제가 완료되면 오버클라우드 스택의 상태를 확인합니다.

```
(undercloud)$ openstack stack list
```

표 16.3. 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.

14. **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>
```

15. **오버클라우드에서 삭제된 Compute 노드를 교체하지 않는 경우 노드 수가 포함된 환경 파일의 ComputeCount 매개변수를 줄입니다. 일반적으로 이 파일의 이름은 node-info.yaml로 지정됩니다. 예를 들어 한 노드를 삭제한 경우 노드 수를 4개의 노드에서 3개의 노드로 줄입니다.**

```
parameter_defaults:
  ...
  ComputeCount: 3
  ...
```

노드 개수를 줄이면 openstack overcloud deploy를 실행할 때 director에서 새 노드를 프로비저닝하지 않습니다.

오버클라우드 배포에서 삭제된 컴퓨팅 노드를 교체하는 경우 삭제된 컴퓨팅 노드 배치를 참조하십시오.

16.3.2. 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. **director**가 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.

16.4. 예측 가능한 IP 주소를 사용하는 노드를 교체할 때 호스트 이름 유지 및 HOSTNAMEMAP

예측 가능한 **IP** 주소를 사용하도록 오버클라우드를 구성하고 **HostNameMap** 을 사용하여 **heat** 기반 호스트 이름을 사전 프로비저닝된 노드의 호스트 이름에 매핑하려면 새 교체 노드 인덱스를 **IP** 주소와 호스트 이름에 매핑하도록 오버클라우드를 구성해야 합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 교체하려는 리소스의 **physical_resource_id** 및 **removed_rsrc_list** 를 검색합니다.

```
(undercloud)$ openstack stack resource show <stack> <role>
```

- **& lt;stack >**을 리소스가 속한 스택의 이름으로 바꿉니다(예: **overcloud**).
- **&lt ;role >**을 노드를 교체할 역할의 이름으로 바꿉니다(예: **Compute**).

출력 예:

```
+-----------------------+-----------------------------------------------------------+
| Field                 | Value                                                     |
+-----------------------+-----------------------------------------------------------+
| attributes            | {u'attributes': None, u'refs': None, u'refs_map': None,   |
|                       | u'removed_rsrc_list': [u'2', u'3']}                       | 1
| creation_time         | 2017-09-05T09:10:42Z                                      |
| description           |                                                           |
| links                 | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9 |
|                       | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-          |
```

```
|                     | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':       |
|                     | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da |
|                     | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e- |
|                     | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h |
|                     | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874 |
|                     | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-       |
|                     | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]        |
| logical_resource_id | Compute                                                  |
| physical_resource_id | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                    |
| required_by         | [u'AllNodesDeploySteps',                                 |
|                     | u'ComputeAllNodesValidationDeployment',                  |
|                     | u'AllNodesExtraConfig', u'ComputeIpListMap',             |
|                     | u'ComputeHostsDeployment', u'UpdateWorkflow',            |
|                     | u'ComputeSshKnownHostsDeployment', u'hostsConfig',       |
|                     | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']    |
| resource_name       | Compute                                                  |
| resource_status     | CREATE_COMPLETE                                          |
| resource_status_reason | state changed                                         |
| resource_type       | OS::Heat::ResourceGroup                                  |
| updated_time        | 2017-09-05T09:10:42Z                                     |
+---------------------+----------------------------------------------------------+
```

1

removed_rsrc_list 는 리소스에 대해 이미 제거된 노드의 인덱스를 나열합니다.

4. **resource_name** 을 검색하여 **heat**가 이 리소스의 노드에 적용한 최대 인덱스를 결정합니다.

```
(undercloud)$ openstack stack resource list <physical_resource_id>
```

- **2**단계에서 검색한 **ID**로 **<physical_resource_id>** 을 바꿉니다.

5. **resource_name** 및 **removed_rsrc_list** 를 사용하여 **heat**가 새 노드에 적용할 다음 인덱스를 확인합니다.

- **removed_rsrc_list** 가 비어 있으면 다음 인덱스가 **(current_maximum_index) + 1**이 됩니다.
- **removed_rsrc_list** 에 값**(current_maximum_index) + 1**이 포함된 경우 다음 인덱스는 사용 가능한 다음 인덱스가 됩니다.

6. 대체 베어 메탈 노드의 **ID**를 검색합니다.

-

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

7. 교체 노드의 기능을 새 인덱스로 업데이트합니다.

```
openstack baremetal node set --property capabilities='node:<role>-<index>,boot_option:local'
<node>
```

- <role>을 노드를 교체할 역할의 이름으로 바꿉니다(예: compute).
- <index>를 5단계에서 계산된 인덱스로 바꿉니다.
- <node>를 베어 메탈 노드의 ID로 바꿉니다.

Compute 스케줄러는 노드 기능을 사용하여 배포의 노드와 일치합니다.

8. HostnameMap 구성에 인덱스를 추가하여 새 노드에 호스트 이름을 할당합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0 (1)
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-controller-3: overcloud-controller-prod-456-0 (2)
    overcloud-compute-0: overcloud-compute-prod-abc-0
    overcloud-compute-3: overcloud-compute-prod-abc-3 (3)
    overcloud-compute-8: overcloud-compute-prod-abc-3 (4)
    ....
```

(1) 제거하고 새 노드로 교체할 노드입니다.

(2) 새 노드.

3

제거하고 새 노드로 교체할 노드입니다.

4

새 노드.

참고

HostnameMap 에서 삭제된 노드의 매핑을 삭제하지 마십시오.

9. 교체 노드의 IP 주소를 네트워크 IP 주소 매핑 파일의 각 네트워크 IP 주소 목록 끝에 추가합니다. **ips-from-pool-all.yaml.** 다음 예에서 새 인덱스의 IP 주소 **overcloud-controller-3** 은 각 **ControllerIPs** 네트워크의 IP 주소 목록 끝에 추가되고 **overcloud-controller-1** 을 대체하므로 **overcloud-controller-1** 과 동일한 IP 주소가 할당됩니다. 새 인덱스의 IP 주소 **overcloud-compute-8** 이 각 **ComputeIPs** 네트워크에 대해 IP 주소 목록 끝에 추가되며 대체 인덱스와 동일한 IP 주소가 할당됩니다. **overcloud-compute-3:**

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10 1
      - 192.168.1.11 2
      - 192.168.1.12 3
      - 192.168.1.11 4
    ...
    storage:
      - 192.168.2.10
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
    ...

  ComputeIPs:
    ...
    internal_api:
      - 172.17.0.10 5
      - 172.17.0.11 6
      - 172.17.0.11 7
    ...
    storage:
      - 172.17.0.10
      - 172.17.0.11
      - 172.17.0.11
    ...
```

1

인덱스 0에 할당된 IP 주소 overcloud-controller-prod-123-0.

2

인덱스 1에 할당된 IP 주소 overcloud-controller-prod-456-0. 이 노드는 인덱스 3으로 교체됩니다. 이 항목을 제거하지 마십시오.

3

인덱스 2에 할당된 IP 주소 overcloud-controller-prod-789-0 호스트 이름입니다.

4

인덱스 3에 할당된 IP 주소, 호스트 이름 overcloud-controller-prod-456-0. 인덱스 1을 대체하는 새 노드입니다.

5

인덱스 0에 할당된 IP 주소 overcloud-compute-0.

6

인덱스 1에 할당된 IP 주소 overcloud-compute-3. 이 노드는 인덱스 2로 교체됩니다. 이 항목을 제거하지 마십시오.

7

인덱스 2에 할당된 IP 주소, 호스트 이름 overcloud-compute-8. 인덱스 1을 대체하는 새 노드입니다.

16.5. 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 Ceph 가이드를 참조하십시오.

16.6. OBJECT STORAGE 노드 교체

이 섹션에서는 클러스터의 무결성에 영향을 주지 않고 Object Storage 노드를 교체하는 방법을 설명합니다. 다음 예제에서는 세 개의 노드로 이루어진 Object Storage 클러스터에서 노드를 overcloud-objectstorage-1 노드를 교체합니다. 이 절차의 목표는 노드를 하나 추가한 다음 overcloud-

objectstorage-1 노드를 삭제하는 것입니다. **overcloud-objectstorage-1** 노드가 새로운 노드로 대체됩니다.

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 예제에서는 **Object Storage** 노드의 수량을 **3**에서 **4**로 확장합니다.

2. 업데이트된 **ObjectStorageCount** 매개변수를 사용하여 배포 명령을 실행합니다.

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates -e node-info.yaml
<environment_files>
```

배포 명령이 완료되면 오버클라우드에 추가 **Object Storage** 노드가 포함됩니다.

3. 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*
```

4. 링에서 이전 노드를 삭제하려면 **ObjectStorageCount** 매개변수를 줄여 이전 노드를 제거합니다. 이 예제에서는 **ObjectStorageCount** 매개변수를 **3**으로 줄입니다.

```
parameter_defaults:
  ObjectStorageCount: 3
```

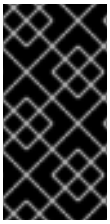
5. **remove-object-node.yaml**이라는 새 환경 파일을 생성합니다. 이 파일은 지정된 **Object Storage** 노드를 식별하고 삭제합니다. 다음 콘텐츠는 **overcloud-objectstorage-1**을 삭제하도록 지정합니다.

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

6. 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.



중요

오버클라우드를 처음 생성할 때의 모든 환경 파일과 옵션을 포함합니다. 여기에는 컴퓨팅 이외의 노드에 대한 동일한 확장 매개변수가 포함됩니다.

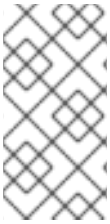
16.7. 건너뛰기 배포 식별자 사용

스택 업데이트 작업 중에 기본적으로 모든 매니페스트를 다시 적용합니다. 이로 인해 시간이 오래 걸릴 수 있으며 이로 인해 작업이 필요하지 않을 수 있습니다.

기본 작업을 재정의하려면 **skip-deploy-identifier** 옵션을 사용합니다.

```
openstack overcloud deploy --skip-deploy-identifier
```

배포 명령이 **DeployIdentifier** 매개변수의 고유 **ID**를 생성하지 않게 하려면 이 옵션을 사용합니다. 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.



참고

puppet 매니페스트 또는 **hierdata**가 변경되면 **--skip-deploy-identifier** 가 지정된 경우에도 **puppet**은 모든 매니페스트를 다시 적용합니다.

16.8. 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 새 노드를 확장하고 기존 노드를 제외하려는 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]
```

heat는 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 서버의 변경 사항을 분리하는 자체 방식이 있습니다.

- 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 **DeploymentServerBlacklist**를 편집하여 빈 배열을 사용합니다.

```
parameter_defaults:
  DeploymentServerBlacklist: []
```

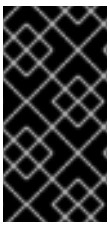
주의

DeploymentServerBlacklist 매개변수를 생략하지 마십시오. 매개변수를 생략하면 오버클라우드 배포에서 이전에 저장된 값을 사용합니다.

17장. 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.

17.1. 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. 데이터베이스 클라이언트 툴을 설치합니다.

```
(undercloud)$ sudo dnf -y install mariadb
```

3. 데이터베이스에 **root** 사용자 액세스를 구성합니다.

```
(undercloud)$ sudo cp /var/lib/config-data/puppet-generated/mysql/root/.my.cnf /root/.
```

4. 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
```

5. 새 노드를 프로비저닝하는 경우 언더클라우드에 이미지 캐싱 및 변환을 수행하는 데 필요한 **10GB**의 가용 스토리지가 있는지 확인합니다.

```
(undercloud)$ df -h
```

6. 새 컨트롤러 노드의 **IP** 주소를 재사용하는 경우 이전 컨트롤러에서 사용하는 포트를 삭제해야 합니다.

```
(undercloud)$ openstack port delete <port>
```

7. 실행 중인 컨트롤러 노드에서 **Pacemaker**의 상태를 확인합니다. 예를 들어 실행 중인 컨트롤러 노드의 **IP** 주소가 **192.168.0.47**인 경우 다음 명령을 사용하여 **Pacemaker** 상태를 확인합니다.

```
(undercloud)$ ssh heat-admin@192.168.0.47 'sudo pcs status'
```

기존 노드에서 실행 중인 모든 서비스와 실패한 노드에서 중지된 모든 서비스가 출력에 표시됩니다.

8. 오버클라우드의 **MariaDB** 클러스터에 있는 각 노드에서 다음 매개변수를 확인합니다.

- **wsrep_local_state_comment:** 동기화됨
- **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 **192.168.0.46**입니다.

```
(undercloud)$ for i in 192.168.0.46 192.168.0.47 ; do echo "*** $i ***" ; ssh heat-
admin@$i "sudo podman exec \$(sudo podman ps --filter name=galera-bundle -q) mysql
```

```
-e \"SHOW STATUS LIKE 'wsrep_local_state_comment'; SHOW STATUS LIKE
'wsrep_cluster_size';\""; done
```

9. **RabbitMQ 상태를 확인합니다. 예를 들어 실행 중인 컨트롤러 노드의 IP 주소가 192.168.0.47인 경우 다음 명령을 사용하여 RabbitMQ 상태를 확인합니다.**

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo podman exec \$(sudo podman ps -f
name=rabbitmq-bundle -q) rabbitmqctl cluster_status"
```

running_nodes 키는 사용 가능한 두 개의 노드만 표시하고, 실패한 노드는 표시하지 않습니다.

10. **펜싱이 활성화된 경우 비활성화합니다. 예를 들어 실행 중인 컨트롤러 노드의 IP 주소가 192.168.0.47이면 다음 명령을 사용하여 펜싱 상태를 확인합니다.**

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
```

펜싱을 비활성화하려면 다음 명령을 실행합니다.

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
```

11. **Compute 서비스가 director 노드에서 활성 상태인지 확인합니다.**

```
(undercloud)$ openstack hypervisor list
```

출력에 모든 유지보수 이외의 모드 노드가 up으로 표시됩니다.

12. **모든 언더클라우드 컨테이너가 실행 중인지 확인합니다.**

```
(undercloud)$ sudo podman ps
```

13. **실패한 컨트롤러 노드에서 실행 중인 nova_* 컨테이너를 모두 중지합니다.**

```
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api_cron.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_compute.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_conductor.service
```

```
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_metadata.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_placement.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_scheduler.service
```

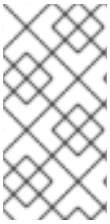
14. 선택 사항: Bare Metal Service(ironic)를 virt 드라이버로 사용하는 경우, 제거 중인 컨트롤러로 instances.host 가 설정된 베어 메탈 인스턴스의 셀 데이터베이스의 서비스 항목을 수동으로 업데이트해야 합니다. Red Hat 지원 센터에 문의하십시오.

참고

베어 메탈 서비스(ironic)를 가상화 드라이버로 사용할 때 셀 데이터베이스를 수동으로 업데이트하는 이 수동 업데이트는 BZ2017980 이 완료될 때까지 노드가 리밸런싱되도록 하는 임시 해결 방법입니다.

17.2. CEPH MONITOR 데몬 삭제

컨트롤러 노드가 Ceph 모니터 서비스를 실행하는 경우 다음 단계를 완료하여 ceph-mon 데몬을 제거합니다.



참고

클러스터에 새 컨트롤러 노드를 추가하면 새 Ceph 모니터 데몬도 자동으로 추가됩니다.

절차

1. 교체할 컨트롤러 노드에 연결하고 root 사용자로 전환합니다.

```
# ssh heat-admin@192.168.0.47
# sudo su -
```

참고

컨트롤러 노드에 연결할 수 없는 경우 1단계와 2단계를 건너뛰고 작동하는 모든 컨트롤러 노드에서 절차 3단계를 계속 진행합니다.

2. 모니터를 중지합니다.

```
# systemctl stop ceph-mon@<monitor_hostname>
```

-

예를 들면 다음과 같습니다.

```
# systemctl stop ceph-mon@overcloud-controller-1
```

3. 교체하려는 컨트롤러 노드에서 연결을 끊습니다.

4. 기존 컨트롤러 노드 중 하나에 연결합니다.

```
# ssh heat-admin@192.168.0.46
# sudo su -
```

5. 클러스터에서 모니터를 삭제합니다.

```
# sudo podman exec -it ceph-mon-controller-0 ceph mon remove overcloud-controller-1
```

6. 모든 컨트롤러 노드에서 **/etc/ceph/ceph.conf**의 **v1** 및 **v2** 모니터 항목을 삭제합니다. 예를 들어 **controller-1**을 삭제하면 **controller-1**의 **IP**와 호스트 이름이 삭제됩니다.

편집 전:

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.22:3300,v1:172.18.0.22:6789],[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-1,overcloud-controller-0
```

편집 후:

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-0
```

참고

교체용 컨트롤러 노드를 추가하면 **director**가 **ceph.conf** 파일을 관련 오버클라우드 노드에서 업데이트합니다. 일반적으로 이 구성 파일은 **director**에서만 관리하며 수동으로 편집해서는 안 됩니다. 그러나 새 노드를 추가하기 전에 다른 노드가 재시작되는 경우 일관성을 보장하기 위해 파일을 수동으로 편집할 수 있습니다.

7. (선택 사항) 모니터 데이터를 압축하고 다른 서버에 압축 파일을 저장할 수 있습니다.

```
# mv /var/lib/ceph/mon/<cluster>-<daemon_id> /var/lib/ceph/mon/removed-<cluster>-
<daemon_id>
```

17.3. 컨트롤러 노드 교체를 위한 클러스터 준비

기존 노드를 교체하기 전에 노드에서 **Pacemaker**가 실행되지 않는지 확인한 후 **Pacemaker** 클러스터에서 해당 노드를 삭제해야 합니다.

절차

1. 컨트롤러 노드의 **IP** 주소 목록을 보려면 다음 명령을 실행합니다.

```
(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 **Pacemaker**를 중지합니다. 이 예제에서는 **overcloud-controller-1**의 **Pacemaker**를 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 **Pacemaker**가 이미 중지되었으므로 위 작업을 수행하지 않아도 됩니다.

3. 기존 노드에서 **Pacemaker**를 중지한 후 **pacemaker** 클러스터에서 기존 노드를 삭제합니다. 다음 예제 명령은 **overcloud-controller-0**에 로그인하여 **overcloud-controller-1**을 삭제합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-
controller-1"
```

-

하드웨어 장애 등으로 인해 교체할 노드에 연결할 수 없는 경우, **--skip-offline** 및 **--force** 추가 옵션으로 **pcs** 명령을 실행하여 클러스터에서 노드를 강제로 삭제합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-controller-1 --skip-offline --force"
```

4.

pacemaker 클러스터에서 기존 노드를 삭제한 후 **pacemaker**의 알려진 호스트 목록에서 노드를 삭제합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs host deauth overcloud-controller-1"
```

노드에 연결할 수 있는지 여부에 관계없이 이 명령을 실행할 수 있습니다.

5.

교체 후 새 컨트롤러 노드에서 올바른 **STONITH** 펜싱 장치를 사용하는지 확인하려면 다음 명령을 입력하여 노드에서 기존 장치를 삭제합니다.

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs stonith delete <stonith_resource_name>"
```

- **<stonith_resource_name>**을 기존 노드에 해당하는 **STONITH** 리소스의 이름으로 바꿉니다. 리소스 이름은 **<resource_agent>-<host_mac>** 포맷을 사용합니다. **fencing.yaml** 파일의 **FencingConfig** 섹션에서 리소스 에이전트 및 호스트 **MAC** 주소를 찾을 수 있습니다.

6.

오버클라우드 데이터베이스는 교체 절차 중 계속 실행되고 있어야 합니다. 다음 절차 중에 **Pacemaker**에서 **Galera**를 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"
```

17.4. 컨트롤러 노드 교체

컨트롤러 노드를 교체하려면 교체할 노드의 인덱스를 확인합니다.

- 노드가 가상 노드인 경우 오류가 발생한 디스크를 포함하는 노드를 확인하고 백업에서 디스크를 복원합니다. 오류가 발생한 서버에서 **PXE** 부팅에 사용되는 **NIC**의 **MAC** 주소가 디스크 교체 후에도 동일하게 유지되는지 확인합니다.

- 노드가 베어 메탈 노드인 경우 디스크를 교체하고, 오버클라우드 구성을 사용하여 새 디스크를 준비한 후 새 하드웨어에서 노드 인트로스펙션을 수행합니다.

- 노드가 펜싱 기능이 있는 고가용성 클러스터에 속하는 경우 **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.

overcloud-controller-1 노드를 **overcloud-controller-3** 노드로 교체하려면 다음 예제 단계를 완료합니다. **overcloud-controller-3** 노드의 **ID**는 **75b25e9a-948d-424a-9b3b-f0ef70a6eacf**입니다.

중요

노드를 기존 베어 메탈 노드로 교체하려면 **director**가 노드를 자동으로 다시 프로비저닝하지 않도록 발신 노드에서 유지보수 모드를 활성화합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. **overcloud-controller-1** 노드의 인덱스를 확인합니다.

```
$ INSTANCE=$(openstack server list --name overcloud-controller-1 -f value -c ID)
```

3. 인스턴스와 연결된 베어 메탈 노드를 확인합니다.

```
$ NODE=$(openstack baremetal node list -f csv --quote minimal | grep $INSTANCE | cut -f1 -d,)
```

4. 노드를 유지보수 모드로 설정합니다.

```
$ openstack baremetal node maintenance set $NODE
```

5. 컨트롤러 노드가 가상 노드인 경우 컨트롤러 호스트에서 다음 명령을 실행하여 백업에 있는 가상 디스크를 교체합니다.

```
$ cp <VIRTUAL_DISK_BACKUP> /var/lib/libvirt/images/<VIRTUAL_DISK>
```

- **<VIRTUAL_DISK_BACKUP>**을 오류가 발생한 가상 디스크의 백업 경로로 교체하고, **<VIRTUAL_DISK>**를 교체할 가상 디스크 이름으로 교체합니다.

  발신 노드 백업이 없는 경우 가상화된 노드를 사용해야 합니다.

  컨트롤러 노드가 베어 메탈 노드인 경우 다음 단계를 완료하여 디스크를 새 베어 메탈 디스크로 교체합니다.

  a. 물리 하드 드라이브 또는 솔리드 스테이트 드라이브를 교체합니다.

  b. 오류가 발생한 노드와 동일한 구성으로 노드를 준비합니다.

6. 연결되지 않은 노드를 나열하고 새 노드의 **ID**를 확인합니다.

```
$ openstack baremetal node list --unassociated
```

7. 새 노드를 **control** 프로필로 태그합니다.

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-9b3b-f0ef70a6eacf
```

17.5. 부트스트랩 컨트롤러 노드 교체

부트스트랩 작업에 사용하는 컨트롤러 노드를 교체하고 노드 이름을 유지하려면 다음 단계를 완료하여 교체 프로세스 후 부트스트랩 컨트롤러 노드의 이름을 설정합니다.

절차

1. 다음 명령을 실행하여 부트스트랩 컨트롤러 노드의 이름을 찾습니다.

```
ssh tripleo-admin@CONTROLLER_IP "sudo hiera -c /etc/puppet/hiera.yaml
pacemaker_short_bootstrap_node_name"
```

- **CONTROLLER_IP**를 활성 컨트롤러 노드의 **IP** 주소로 바꿉니다.

2. 환경 파일에 **ExtraConfig** 섹션이 포함되어 있는지 확인합니다. **ExtraConfig** 매개변수가 없는 경우 **~/templates/bootstrap-controller.yaml** 환경 파일을 생성하고 다음 콘텐츠를 추가합니다.

```
parameter_defaults:
  ExtraConfig:
    pacemaker_short_bootstrap_node_name: NODE_NAME
    mysql_short_bootstrap_node_name: NODE_NAME
```

- **NODE_NAME**을 교체 프로세스 후 부트스트랩 작업에 사용하려는 기존 컨트롤러 노드의 이름으로 바꿉니다.

환경 파일에 **ExtraConfig** 매개변수가 이미 포함된 경우 **pacemaker_short_bootstrap_node_name** 및 **mysql_short_bootstrap_node** 매개변수를 설정하는 행만 추가합니다.

3. 단계를 수행하여 컨트롤러 노드 교체를 트리거하고 **overcloud deploy** 명령에 환경 파일을 포함합니다. 자세한 내용은 컨트롤러 노드 교체 트리거를 참조하십시오.

부트스트랩 컨트롤러 노드 교체 문제 해결에 대한 정보는 **Replacement of the first Controller node fails at step 1 if the same hostname is used for a new node** 문서를 참조하십시오.

17.6. 예측 가능한 IP 주소를 사용하는 노드를 교체할 때 호스트 이름 유지 및 HOSTNAMEMAP

예측 가능한 **IP** 주소를 사용하도록 오버클라우드를 구성하고 **HostNameMap** 을 사용하여 **heat** 기반 호스트 이름을 사전 프로비저닝된 노드의 호스트 이름에 매핑하려면 새 교체 노드 인덱스를 **IP** 주소와 호스트 이름에 매핑하도록 오버클라우드를 구성해야 합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 교체하려는 리소스의 **physical_resource_id** 및 **removed_rsrc_list** 를 검색합니다.

```
(undercloud)$ openstack stack resource show <stack> <role>
```

- **& lt;stack >**을 리소스가 속한 스택의 이름으로 바꿉니다(예: **overcloud**).
- **&lt ;role >**을 노드를 교체할 역할의 이름으로 바꿉니다(예: **Compute**).

출력 예:

```
+-----------------------+-----------------------------------------------------------+
| Field                 | Value                                                     |
+-----------------------+-----------------------------------------------------------+
| attributes            | {u'attributes': None, u'refs': None, u'refs_map': None,   |
|                       | u'removed_rsrc_list': [u'2', u'3']}                       | 1
| creation_time         | 2017-09-05T09:10:42Z                                      |
| description           |                                                           |
| links                 | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9 |
|                       | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-          |
|                       | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':        |
|                       | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da |
|                       | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e- |
|                       | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h |
|                       | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874 |
|                       | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-        |
|                       | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]         |
| logical_resource_id   | Compute                                                   |
| physical_resource_id  | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                      |
| required_by           | [u'AllNodesDeploySteps',                                  |
|                       | u'ComputeAllNodesValidationDeployment',                   |
|                       | u'AllNodesExtraConfig', u'ComputeIpListMap',              |
|                       | u'ComputeHostsDeployment', u'UpdateWorkflow',             |
|                       | u'ComputeSshKnownHostsDeployment', u'hostsConfig',        |
|                       | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']     |
| resource_name         | Compute                                                   |
| resource_status       | CREATE_COMPLETE                                           |
| resource_status_reason | state changed                                            |
| resource_type         | OS::Heat::ResourceGroup                                   |
| updated_time          | 2017-09-05T09:10:42Z                                      |
+-----------------------+-----------------------------------------------------------+
```

1 **removed_rsrc_list** 는 리소스에 대해 이미 제거된 노드의 인덱스를 나열합니다.

4. **resource_name** 을 검색하여 **heat**가 이 리소스의 노드에 적용한 최대 인덱스를 결정합니다.

```
(undercloud)$ openstack stack resource list <physical_resource_id>
```

- **2**단계에서 검색한 **ID**로 **<physical_resource_id>** 을 바꿉니다.

5. **resource_name** 및 **removed_rsrc_list** 를 사용하여 **heat**가 새 노드에 적용할 다음 인덱스를 확인합니다.

- **removed_rsrc_list** 가 비어 있으면 다음 인덱스가 **(current_maximum_index) + 1**이 됩니다.

- **removed_rsrc_list** 에 값**(current_maximum_index) + 1**이 포함된 경우 다음 인덱스는 사용 가능한 다음 인덱스가 됩니다.

6. 대체 베어 메탈 노드의 **ID**를 검색합니다.

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

7. 교체 노드의 기능을 새 인덱스로 업데이트합니다.

```
openstack baremetal node set --property capabilities='node:<role>-<index>,boot_option:local'
<node>
```

- **&lt ;role >**을 노드를 교체할 역할의 이름으로 바꿉니다**(**예**: compute).**

- **& lt;index& gt;**를 **5**단계에서 계산된 인덱스로 바꿉니다.

- **& lt;node& gt;**를 베어 메탈 노드의 **ID**로 바꿉니다.

Compute 스케줄러는 노드 기능을 사용하여 배포의 노드와 일치합니다.

8. **HostnameMap** 구성에 인덱스를 추가하여 새 노드에 호스트 이름을 할당합니다. 예를 들면

다음과 같습니다.

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0 (1)
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-controller-3: overcloud-controller-prod-456-0 (2)
    overcloud-compute-0: overcloud-compute-prod-abc-0
    overcloud-compute-3: overcloud-compute-prod-abc-3 (3)
    overcloud-compute-8: overcloud-compute-prod-abc-3 (4)
    ....
```

(1) 제거하고 새 노드로 교체할 노드입니다.

(2) 새 노드.

(3) 제거하고 새 노드로 교체할 노드입니다.

(4) 새 노드.

참고

HostnameMap 에서 삭제된 노드의 매핑을 삭제하지 마십시오.

9. 교체 노드의 **IP** 주소를 네트워크 **IP** 주소 매핑 파일의 각 네트워크 **IP** 주소 목록 끝에 추가합니다. **ips-from-pool-all.yaml.** 다음 예에서 새 인덱스의 **IP** 주소 **overcloud-controller-3** 은 각 **ControllerIPs** 네트워크의 **IP** 주소 목록 끝에 추가되고 **overcloud-controller-1** 을 대체하므로 **overcloud-controller-1** 과 동일한 **IP** 주소가 할당됩니다. 새 인덱스의 **IP** 주소 **overcloud-compute-8** 이 각 **ComputeIPs** 네트워크에 대해 **IP** 주소 목록 끝에 추가되며 대체 인덱스와 동일한 **IP** 주소가 할당됩니다. **overcloud-compute-3:**

```
parameter_defaults:
```

```
ControllerIPs:
  ...
  internal_api:
    - 192.168.1.10 1
    - 192.168.1.11 2
    - 192.168.1.12 3
    - 192.168.1.11 4
  ...
  storage:
    - 192.168.2.10
    - 192.168.2.11
    - 192.168.2.12
    - 192.168.2.11
  ...

ComputeIPs:
  ...
  internal_api:
    - 172.17.0.10 5
    - 172.17.0.11 6
    - 172.17.0.11 7
  ...
  storage:
    - 172.17.0.10
    - 172.17.0.11
    - 172.17.0.11
  ...
```

1

인덱스 **0**에 할당된 **IP** 주소 **overcloud-controller-prod-123-0.**

2

인덱스 **1**에 할당된 **IP** 주소 **overcloud-controller-prod-456-0.** 이 노드는 인덱스 **3**으로 교체됩니다. 이 항목을 제거하지 마십시오.

3

인덱스 **2**에 할당된 **IP** 주소 **overcloud-controller-prod-789-0** 호스트 이름입니다.

4

인덱스 **3**에 할당된 **IP** 주소, 호스트 이름 **overcloud-controller-prod-456-0.** 인덱스 **1**을 대체하는 새 노드입니다.

5

인덱스 **0**에 할당된 **IP** 주소 **overcloud-compute-0.**

6

인덱스 1에 할당된 IP 주소 overcloud-compute-3. 이 노드는 인덱스 2로 교체됩니다. 이 항목을 제거하지 마십시오.

7

인덱스 2에 할당된 IP 주소, 호스트 이름 overcloud-compute-8. 인덱스 1을 대체하는 새 노드입니다.

17.7. 컨트롤러 노드 교체 트리거

기존 컨트롤러 노드를 삭제하고 새 컨트롤러 노드로 교체하려면 다음 단계를 완료합니다.

절차

1. 삭제하려는 컨트롤러 노드의 UUID를 확인하고 < NODEID> 변수에 저장합니다. <node_name> 을 삭제하려는 노드 이름으로 교체해야 합니다.

```
(undercloud)[stack@director ~]$ NODEID=$(openstack server list -f value -c ID --name
<node_name>)
```

2. Heat 리소스 ID를 식별하려면 다음 명령을 입력합니다.

```
(undercloud)[stack@director ~]$ 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. 오버클라우드 배포 명령을 입력하고 remove-controller.yaml 환경 파일 및 해당 환경과 관련된 기타 환경 파일을 포함합니다.

```
(undercloud) $ openstack overcloud deploy --templates \
    -e /home/stack/templates/remove-controller.yaml \
    [OTHER OPTIONS]
```

참고

- 배포 명령의 이 인스턴스에만 **-e ~/templates/remove-controller.yaml**을 포함합니다. 이후의 배포 작업에서는 이 환경 파일을 삭제합니다.

- 부트스트랩 컨트롤러 노드를 교체하고 노드 이름을 유지하려면 **~/templates/bootstrap-controller.yaml**을 포함합니다. 자세한 내용은 부트스트랩 컨트롤러 노드 교체를 참조하십시오.

5. **director**에서 기존 노드를 삭제하고, 새 노드를 생성한 후 오버클라우드 스택을 업데이트합니다. 다음 명령을 사용하여 오버클라우드 스택의 상태를 확인할 수 있습니다.

```
(undercloud)$ openstack stack list --nested
```

6. 배포 명령이 완료되면 기존 노드가 새 노드로 교체되었는지 확인합니다.

```
(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.

17.8. 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 **Galera**를 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. **director**를 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** 에이전트 호스트에 라우터를 추가합니다. 다음 예제 명령을 사용하여 **UUID**가 **2d1c1dc1-d9d4-4fa9-b2c8-f29cd1a649d4**인 **L3** 에이전트에 **r1**이라는 라우터를 추가합니다.

```
(overcloud) $ openstack network agent add router --l3 2d1c1dc1-d9d4-4fa9-b2c8-
f29cd1a649d4 r1
```

9. **cinder** 서비스를 정리합니다.

   a. **cinder** 서비스를 나열합니다.

   ```
   (overcloud) $ openstack volume service list
   ```

   b. 컨트롤러 노드에 로그인하고 **cinder-api** 컨테이너에 연결하고 **cinder-manage service remove** 명령을 사용하여 남은 서비스를 제거합니다.

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
   service remove cinder-backup <host>
   [heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
   service remove cinder-scheduler <host>
   ```

10. **RabbitMQ** 클러스터를 정리합니다.

    a. 컨트롤러 노드에 로그인합니다.

    b. **podman exec** 명령을 사용하여 **bash**를 시작하고 **RabbitMQ** 클러스터의 상태를 확인합니다.

    ```
    [heat-admin@overcloud-controller-0 ~]$ podman exec -it rabbitmq-bundle-podman-0
    bash
    [heat-admin@overcloud-controller-0 ~]$ rabbitmqctl cluster_status
    ```

    c. **rabbitmqctl** 명령을 사용하여 교체된 컨트롤러 노드를 잊어버십시오.

    ```
    [heat-admin@overcloud-controller-0 ~]$ rabbitmqctl forget_cluster_node <node_name>
    ```

11. 부트스트랩 컨트롤러 노드를 교체한 경우 교체 프로세스 후에 환경 파일 **~/templates/bootstrap-controller.yaml**을 제거하거나 기존 환경 파일에서 **pacemaker_short_bootstrap_node_name** 및 **mysql_short_bootstrap_node_name** 매개변수를 삭제해야 합니다. 이 단계에서는 **director**가 후속 교체에서 컨트롤러 노드 이름을 재정의하지 않습니다. 자세한 내용은 부트스트랩 컨트롤러 노드 교체를 참조하십시오.

18장. 노드 재부팅

언더클라우드와 오버클라우드에서 노드를 재부팅해야 하는 경우가 있습니다. 다음 절차에 따라 다양한 노드 유형을 재부팅하는 방법을 알아보십시오.

- 한 역할에 있는 모든 노드를 재부팅하는 경우 각 노드를 하나씩 재부팅하는 것이 좋습니다. 역할의 모든 노드를 동시에 재부팅하면 재부팅 작업 중에 서비스 다운 타임이 발생할 수 있습니다.
- **OpenStack Platform** 환경의 노드를 모두 재부팅하는 경우 다음 순서대로 노드를 재부팅합니다.

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**) 노드를 재부팅합니다.
6. 컴퓨팅 노드 재부팅

18.1. 언더클라우드 노드 재부팅

언더클라우드 호스트를 재부팅하려면 다음 단계를 완료합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. 언더클라우드를 재부팅합니다.

```
$ sudo reboot
```

3. 노드가 부팅될 때까지 기다립니다.

18.2. 컨트롤러 노드 및 구성 가능 노드 재부팅

구성 가능 역할을 기반으로 컨트롤러 노드 및 독립 실행형 노드를 재부팅하고 컴퓨팅 노드 및 **Ceph Storage** 노드를 제외합니다.

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.

검증

1. 서비스가 활성화되었는지 확인합니다.

   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. 노드에서 컨테이너화된 서비스를 사용하는 경우 노드의 모든 컨테이너가 활성화되었는지 확인합니다.

```
[heat-admin@overcloud-controller-0 ~]$ sudo podman ps
```

18.3. 독립형 CEPH MON 노드 재부팅

독립형 **Ceph MON** 노드를 재부팅하려면 다음 단계를 완료합니다.

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.

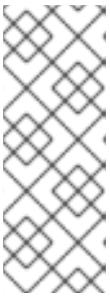
18.4. CEPH STORAGE(OSD) 클러스터 재부팅

Ceph Storage(OSD) 노드 클러스터를 재부팅하려면 다음 단계를 완료합니다.

절차

1. **Ceph MON** 또는 컨트롤러 노드에 로그인하고 **Ceph Storage** 클러스터 재조정을 일시적으로 비활성화합니다.

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
```



참고

다중 스택 또는 **DCN(Distributed Compute node)** 아키텍처가 있는 경우 **noout** 및 **no rebalance** 플래그를 설정할 때 클러스터 이름을 지정해야 합니다. 예: **sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>**

2. 재부팅할 첫 번째 **Ceph Storage** 노드를 선택하고 노드에 로그인합니다.

3. 노드를 재부팅합니다.

```
$ sudo reboot
```

4. 노드가 부팅될 때까지 기다립니다.

5. 노드에 로그인하고 클러스터 상태를 확인합니다.

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

pgmap이 모든 **pgs**를 정상**(active+clean)**으로 보고하는지 확인합니다.

6. 노드에서 로그아웃하고, 다음 노드를 재부팅한 후 상태를 확인합니다. 모든 **Ceph Storage** 노드를 재부팅할 때까지 이 프로세스를 반복합니다.

7. 완료되면 **Ceph MON** 또는 컨트롤러 노드에 로그인하고 클러스터 재조정을 다시 활성화합니다.

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
```

참고

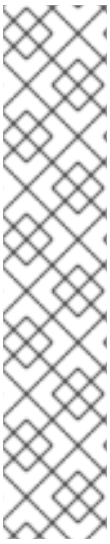
다중 스택 또는 **DCN(Distributed Compute node)** 아키텍처가 있는 경우 **noout** 및 **no rebalance** 플래그를 설정 해제할 때 클러스터 이름을 지정해야 합니다. 예: **sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>**

8. 최종 상태 검사를 수행하여 클러스터가 **HEALTH_OK**를 보고하는지 확인합니다.

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

18.5. 컴퓨팅 노드 재부팅

Red Hat OpenStack Platform 환경에서 인스턴스 다운 타임을 최소화할 수 있도록 인스턴스 마이그레이션 워크플로 에서는 재부팅하려는 컴퓨팅 노드에서 인스턴스를 마이그레이션하기 위해 완료해야 하는 단계를 간략하게 설명합니다.



참고

소스 컴퓨팅 노드에서 다른 컴퓨팅 노드로 인스턴스를 마이그레이션하지 않으면 소스 컴퓨팅 노드에서 인스턴스가 다시 시작될 수 있으므로 업그레이드에 실패할 수 있습니다. 이는 **Podman** 및 **libvirt** 서비스의 변경과 관련된 알려진 문제와 관련이 있습니다.

- **BZ#2009106 - tripleo_nova_libvirt가 두 번 다시 시작한 후 podman panic**
- **BZ#2010135 - tripleo_nova_libvirt가 두 번 다시 시작한 후 podman panic**

인스턴스 워크플로 마이그레이션

1. 노드를 재부팅하기 전에 인스턴스를 다른 컴퓨팅 노드로 마이그레이션할지 여부 결정
2. 새 인스턴스를 프로비저닝하지 않도록 재부팅할 컴퓨팅 노드를 선택하고 비활성화합니다.
3. 인스턴스를 다른 컴퓨팅 노드로 마이그레이션

4. 빈 컴퓨팅 노드 재부팅

5. 빈 컴퓨팅 노드 활성화

사전 요구 사항

- 컴퓨팅 노드를 재부팅하기 전에 노드가 재부팅되는 동안 인스턴스를 다른 컴퓨팅 노드로 마이그레이션할지 여부를 결정해야합니다.

  컴퓨팅 노드 간에 가상 머신 인스턴스를 마이그레이션할 때 발생할 수 있는 마이그레이션 제약 조건 목록을 검토하십시오. 자세한 내용은 ***Configuring the Compute Service for Instance Creation***에서 마이그레이션 제한 조건을 참조하십시오.

- 인스턴스를 마이그레이션할 수 없는 경우 다음과 같은 코어 템플릿 매개변수를 설정하여 컴퓨팅 노드를 재부팅한 후의 인스턴스 상태를 제어할 수 있습니다.

  NovaResumeGuestsStateOnHostBoot

  재부팅한 후에 컴퓨팅 노드에서 인스턴스를 동일한 상태로 되돌릴지 여부를 결정합니다. **False**로 설정하면 인스턴스가 다운된 상태로 유지되며 수동으로 시작해야 합니다. 기본값은 **False** 입니다.

  NovaResumeGuestsShutdownTimeout

  재부팅하기 전에 인스턴스가 종료될 때까지 대기하는 시간(초)입니다. 이 값을 **0**으로 설정하지 않는 것이 좋습니다. 기본값은 **300** 입니다.

  오버클라우드 매개변수 및 사용법에 관한 자세한 내용은 **Overcloud Parameters**를 참조하십시오.

  절차

  1. **stack** 사용자로 언더클라우드에 로그인합니다.

  2. 모든 컴퓨팅 노드 및 해당 **UUID**를 나열합니다.

     ```
     $ source ~/stackrc
     (undercloud) $ openstack server list --name compute
     ```

재부팅할 컴퓨팅 노드의 **UUID**를 확인합니다.

3. 언더클라우드에서 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. 선택 사항: 인스턴스를 다른 컴퓨팅 노드로 마이그레이션하려면 다음 단계를 완료합니다.

a. 인스턴스를 다른 컴퓨팅 노드로 마이그레이션하려면 다음 명령 중 하나를 사용합니다.

- 인스턴스를 다른 호스트로 마이그레이션하려면 다음 명령을 실행합니다.

```
(overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
```

- **nova-scheduler**에서 대상 호스트를 자동으로 선택하도록 합니다.

```
(overcloud) $ nova live-migration <instance_id>
```

- 한 번에 모든 인스턴스를 실시간 마이그레이션합니다.

```
$ nova host-evacuate-live <hostname>
```

참고

nova 명령으로 인해 몇 가지 사용 중단 경고가 표시될 수 있으며, 이러한 경고는 무시해도 됩니다.

b. 마이그레이션이 완료될 때까지 기다립니다.

c. 마이그레이션을 성공적으로 완료했음을 확인합니다.

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

d. 컴퓨팅 노드에 남은 항목이 없을 때까지 인스턴스를 계속 마이그레이션합니다.

6. 컴퓨팅 노드에 로그인하고 노드를 재부팅합니다.

```
[heat-admin@overcloud-compute-0 ~]$ sudo reboot
```

7. 노드가 부팅될 때까지 기다립니다.

8. 컴퓨팅 노드를 다시 활성화합니다.

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname>  nova-compute --enable
```

9. 컴퓨팅 노드가 활성화되었는지 확인합니다.

```
(overcloud) $ openstack compute service list
```

19장. 언더클라우드 및 오버클라우드 종료 및 시작

언더클라우드 및 오버클라우드에서 유지보수를 수행해야 하는 경우 오버클라우드를 시작할 때 최소한의 문제를 확인하려면 특정 순서로 언더클라우드 및 오버클라우드 노드를 종료하고 시작해야 합니다.

사전 요구 사항

- 실행 중인 언더클라우드와 오버클라우드

19.1. 언더클라우드 및 오버클라우드 종료 순서

Red Hat OpenStack Platform 환경을 종료하려면 다음 순서로 오버클라우드 및 언더클라우드를 종료해야 합니다.

1. 오버클라우드 컴퓨팅 노드에서 인스턴스 종료
2. 컴퓨팅 노드 종료
3. 컨트롤러 노드에서 고가용성 및 **OpenStack Platform** 서비스를 모두 중지합니다.
4. **Ceph Storage** 노드 종료
5. 컨트롤러 노드 종료
6. 언더클라우드 종료

19.2. 오버클라우드 컴퓨팅 노드에서 인스턴스 종료

Red Hat OpenStack Platform 환경 종료 과정의 일환으로 **Compute** 노드를 종료하기 전에 컴퓨팅 노드의 모든 인스턴스를 종료합니다.

사전 요구 사항

- 활성 **Compute** 서비스가 있는 오버클라우드

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. 오버클라우드의 인증 정보 파일을 가져옵니다.

```
$ source ~/overcloudrc
```

3. 오버클라우드에서 실행 중인 인스턴스를 확인합니다.

```
$ openstack server list --all-projects
```

4. 오버클라우드에서 각 인스턴스를 중지합니다.

```
$ openstack server stop <INSTANCE>
```

오버클라우드의 모든 인스턴스를 중지할 때까지 각 인스턴스에 대해 이 단계를 반복합니다.

19.3. 컴퓨팅 노드 종료

Red Hat OpenStack Platform 환경 종료 과정의 일환으로 각 컴퓨팅 노드에 로그인하여 종료합니다.

사전 요구 사항

- 컴퓨팅 노드에서 모든 인스턴스 종료

절차

1. 컴퓨팅 노드에 **root** 사용자로 로그인합니다.
2. 노드를 종료합니다.

```
# shutdown -h now
```

3. 모든 컴퓨팅 노드를 종료할 때까지 각 컴퓨팅 노드에 대해 다음 단계를 수행합니다.

19.4. 컨트롤러 노드에서 서비스 중지

Red Hat OpenStack Platform 환경 종료 과정의 일환으로 컨트롤러 노드에서 서비스를 중지한 후 노드를 종료합니다. 여기에는 Pacemaker 및 systemd 서비스가 포함됩니다.

사전 요구 사항

- 활성 Pacemaker 서비스가 있는 오버클라우드

절차

1. root 사용자로 컨트롤러 노드에 로그인합니다.

2. Pacemaker 클러스터를 중지합니다.

```
# pcs cluster stop --all
```

이 명령은 모든 노드에서 클러스터를 중지합니다.

3. Pacemaker 서비스가 중지될 때까지 기다린 후 서비스가 중지되었는지 확인합니다.

   a. Pacemaker 상태를 확인합니다.

   ```
   # pcs status
   ```

   b. Podman에서 Pacemaker 서비스가 실행되고 있지 않은지 확인합니다.

   ```
   # podman ps --filter "name=.*-bundle.*"
   ```

4. Red Hat OpenStack Platform 서비스를 중지합니다.

```
# systemctl stop 'tripleo_*'
```

5. 서비스가 중지될 때까지 기다린 후 **Podman**에서 서비스가 더 이상 실행되지 않는지 확인합니다.

```
# podman ps
```

19.5. CEPH STORAGE 노드 종료

Red Hat OpenStack Platform 환경 종료 과정의 일환으로 **Ceph Storage** 서비스를 비활성화한 다음 각 **Ceph Storage** 노드에 로그인하여 종료합니다.

사전 요구 사항

- 정상적인 **Ceph Storage** 클러스터
- **Ceph MON** 서비스는 독립 실행형 **Ceph MON** 노드 또는 컨트롤러 노드에서 실행 중입니다.

절차

1. 컨트롤러 노드 또는 독립 실행형 **Ceph MON** 노드와 같은 **Ceph MON** 서비스를 실행하는 노드에 **root** 사용자로 로그인합니다.

2. 클러스터 상태를 확인합니다. 다음 예제에서 **podman** 명령은 컨트롤러 노드의 **Ceph MON** 컨테이너 내에서 상태 점검을 실행합니다.

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

상태가 **HEALTH_OK**인지 확인합니다.

3. 클러스터에 **noout, norecover, norebalance, nobackfill, nodown, pause** 플래그를 설정합니다. 다음 예제에서 **podman** 명령은 컨트롤러 노드의 **Ceph MON** 컨테이너를 통해 이러한 플래그를 설정합니다.

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
```

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd set pause
```

4. 각 **Ceph Storage** 노드를 종료합니다.

   a. **root** 사용자로 **Ceph Storage** 노드에 로그인합니다.

   b. 노드를 종료합니다.

   ```
   # shutdown -h now
   ```

   c. 모든 **Ceph Storage** 노드를 종료할 때까지 각 **Ceph Storage** 노드에 대해 다음 단계를 수행합니다.

5. 독립 실행형 **Ceph MON** 노드를 종료합니다.

   a. 독립 실행형 **Ceph MON** 노드에 **root** 사용자로 로그인합니다.

   b. 노드를 종료합니다.

   ```
   # shutdown -h now
   ```

   c. 독립 실행형 **Ceph MON** 노드를 모두 종료할 때까지 각 독립 실행형 **Ceph MON** 노드에 대해 다음 단계를 수행합니다.

추가 리소스

- "시스템을 종료하고 전체 ceph 클러스터를 가져오는 절차는 무엇입니까?"

19.6. 컨트롤러 노드 종료

Red Hat OpenStack Platform 환경 종료 과정의 일환으로 각 컨트롤러 노드에 로그인하여 종료합니다.

사전 요구 사항

- **Pacemaker** 클러스터 중지
- 컨트롤러 노드에서 모든 **Red Hat OpenStack Platform** 서비스를 중지합니다.

절차

1. **root** 사용자로 컨트롤러 노드에 로그인합니다.
2. 노드를 종료합니다.

```
# shutdown -h now
```

3. 모든 컨트롤러 노드를 종료할 때까지 각 컨트롤러 노드에 대해 다음 단계를 수행합니다.

19.7. 언더클라우드 종료

Red Hat OpenStack Platform 환경 종료 과정의 일환으로 언더클라우드 노드에 로그인하여 언더클라우드를 종료합니다.

사전 요구 사항

- 실행 중인 언더클라우드

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. 언더클라우드를 종료합니다.

```
$ sudo shutdown -h now
```

19.8. 시스템 유지보수 수행

언더클라우드와 오버클라우드를 완전히 종료한 후 해당 환경의 시스템에 대한 유지 관리를 수행한 다음 언더클라우드 및 오버클라우드를 시작합니다.

19.9. 언더클라우드 및 오버클라우드 시작 순서

Red Hat OpenStack Platform 환경을 시작하려면 다음 순서로 언더클라우드 및 오버클라우드를 시작해야 합니다.

1. 언더클라우드 시작
2. 컨트롤러 노드 시작
3. **Ceph Storage** 노드 시작
4. 컴퓨팅 노드 시작
5. 오버클라우드 컴퓨팅 노드에서 인스턴스 시작

19.10. 언더클라우드 시작

Red Hat OpenStack Platform 환경 시작 과정의 일환으로 언더클라우드 노드의 전원을 켜고 언더클라우드에 로그인한 다음, 언더클라우드 서비스를 확인합니다.

사전 요구 사항

- 전원이 꺼진 언더클라우드

절차

1. 언더클라우드의 전원을 켜고 언더클라우드가 부팅될 때까지 기다립니다.

검증

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. 언더클라우드에서 서비스를 확인합니다.

```
$ systemctl list-units 'tripleo_*'
```

3. 언더클라우드의 인증 정보 파일을 가져오고 검증 명령을 실행하여 모든 서비스 및 컨테이너가 활성 상태이고 정상 상태인지 확인합니다.

```
$ source stackrc
$ openstack tripleo validator run --validation service-status --limit undercloud
```

추가 리소스

- 검증 프레임워크 사용

19.11. 컨트롤러 노드 시작

Red Hat OpenStack Platform 환경 시작 과정의 일환으로 각 컨트롤러 노드의 전원을 켜고 노드에서 **Pacemaker**가 아닌 서비스를 확인합니다.

사전 요구 사항

- 전원이 꺼진 컨트롤러 노드

절차

1. 각 컨트롤러 노드의 전원을 켭니다.

검증

1. 각 컨트롤러 노드에 **root** 사용자로 로그인합니다.

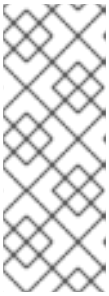
2. 컨트롤러 노드에서 서비스를 확인합니다.

```
$ systemctl -t service
```

Pacemaker 기반이 아닌 서비스만 실행 중입니다.

3. **Pacemaker** 서비스가 시작될 때까지 기다렸다가 서비스가 시작되었는지 확인합니다.

```
$ pcs status
```



참고

환경에서 인스턴스 **HA**를 사용하는 경우 **Compute** 노드를 시작하거나 **pcs stonith confirm <compute_node>** 명령을 사용하여 수동 언펜스 작업을 수행할 때까지 **Pacemaker** 리소스가 시작되지 않습니다. 인스턴스 **HA**를 사용하는 각 컴퓨팅 노드에서 이 명령을 실행해야 합니다.

19.12. CEPH STORAGE 노드 시작

Red Hat OpenStack Platform 환경 시작 과정의 일환으로 **Ceph MON** 및 **Ceph Storage** 노드의 전원을 켜고 **Ceph Storage** 서비스를 활성화합니다.

사전 요구 사항

- 전원이 꺼진 **Ceph Storage** 클러스터
- **Ceph MON** 서비스는 전원이 꺼진 독립 실행형 **Ceph MON** 노드 또는 전원이 켜진 컨트롤러 노드에서 활성화됩니다.

절차

1. 환경에 독립 실행형 **Ceph MON** 노드가 있는 경우 각 **Ceph MON** 노드의 전원을 켭니다.
2. 각 **Ceph Storage** 노드의 전원을 켭니다.
3. 컨트롤러 노드 또는 독립 실행형 **Ceph MON** 노드와 같은 **Ceph MON** 서비스를 실행하는 노드에 **root** 사용자로 로그인합니다.
4. 클러스터 노드의 상태를 확인합니다. 다음 예제에서 **podman** 명령은 컨트롤러 노드의 **Ceph**

MON 컨테이너 내에서 상태 점검을 실행합니다.

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

각 노드의 전원이 켜져 있고 연결되어 있는지 확인합니다.

5. 클러스터의 **noout, norecover, norebalance, nobackfill, nodown** 및 **pause** 플래그를 설정 해제합니다. 다음 예제에서 **podman** 명령은 컨트롤러 노드의 **Ceph MON** 컨테이너를 통해 이러한 플래그를 설정 해제합니다.

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset pause
```

검증

1. 클러스터 상태를 확인합니다. 다음 예제에서 **podman** 명령은 컨트롤러 노드의 **Ceph MON** 컨테이너 내에서 상태 점검을 실행합니다.

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

상태가 **HEALTH_OK**인지 확인합니다.

추가 리소스

- "시스템을 종료하고 전체 **ceph** 클러스터를 가져오는 절차는 무엇입니까?"

19.13. 컴퓨팅 노드 시작

Red Hat OpenStack Platform 환경 시작 과정의 일환으로 각 컴퓨팅 노드의 전원을 켜고 노드에서 서비스를 확인합니다.

사전 요구 사항

- 전원이 꺼진 컴퓨팅 노드

절차

1. 각 컴퓨팅 노드의 전원을 켭니다.

검증

1. 각 **Compute** 노드에 **root** 사용자로 로그인합니다.

2. 컴퓨팅 노드에서 서비스를 확인합니다.

```
$ systemctl -t service
```

19.14. 오버클라우드 컴퓨팅 노드에서 인스턴스 시작

Red Hat OpenStack Platform 환경을 시작하는 과정의 일환으로 컴퓨팅 노드에서 인스턴스를 시작합니다.

사전 요구 사항

- 활성 노드가 있는 활성 오버클라우드

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. 오버클라우드의 인증 정보 파일을 가져옵니다.

```
$ source ~/overcloudrc
```

3. 오버클라우드에서 실행 중인 인스턴스를 확인합니다.

```
$ openstack server list --all-projects
```

4. 오버클라우드에서 인스턴스를 시작합니다.

```
$ openstack server start <INSTANCE>
```

20장. 사용자 지정 SSL/TLS 인증서 설정

공용 끝점에서의 통신에 SSL/TLS를 사용하도록 언더클라우드를 수동으로 설정할 수 있습니다. SSL/TLS를 사용하여 언더클라우드 끝점을 수동으로 설정하면 개념 증명으로 보안 엔드 포인트를 생성합니다. Red Hat은 인증 기관 솔루션을 사용하는 것이 좋습니다.

CA(인증 기관) 솔루션을 사용하는 경우 인증서 갱신, 인증서 해지 목록(CRL) 및 업계 허용 암호화와 같은 프로덕션 준비 솔루션이 있습니다. Red Hat IdM(Identity Manager)을 CA로 사용하는 방법에 대한 자세한 내용은 Ansible을 사용하여 TLS-e 구현을 참조하십시오.

고유한 인증 기관의 SSL 인증서를 사용하려면 다음 설정 단계를 완료해야 합니다.

20.1. 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.

절차

1. /etc/pki/CA/index.txt 파일에는 서명된 모든 인증서의 기록이 포함되어 있습니다. 파일 시스템 경로와 index.txt 파일이 있는지 확인합니다.

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. /etc/pki/CA/serial 파일은 서명할 다음 인증서에 사용할 다음 일련번호를 식별합니다. 이 파일이 있는지 확인합니다. 파일이 없는 경우 새 시작 값으로 새 파일을 생성합니다.

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

20.2. 인증 기관 생성

일반적으로는 외부 인증 기관을 통해 SSL/TLS 인증서에 서명합니다. 고유한 인증 기관을 사용하려는 경우도 있습니다. 예를 들어 내부 전용 인증 기관을 사용할 수도 있습니다.

절차

1. 인증 기관 역할을 하는 키와 인증서 쌍을 생성합니다.

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

2. **openssl req** 명령은 기관에 대한 특정 세부 정보를 요청합니다. 메시지가 나타나면 해당 세부 정보를 입력합니다. 이 명령을 수행하면 **ca.crt.pem**이라는 인증 기관 파일이 생성됩니다.

3. **enable-tls.yaml** 파일에서 인증서 위치를 **PublicTLSCAFile** 매개변수 값으로 설정합니다. 인증서 위치를 **PublicTLSCAFile** 매개변수 값으로 설정하면 **CA** 인증서 경로가 **clouds.yaml** 인증 파일에 추가됩니다.

```
parameter_defaults:
  PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

20.3. 클라이언트에 인증 기관 추가

외부 클라이언트가 **SSL/TLS**를 사용하여 통신하려는 경우 **Red Hat OpenStack Platform** 환경에 액세스해야 하는 각 클라이언트에 인증 기관 파일을 복사합니다.

절차

1. 클라이언트 시스템에 인증 기관을 복사합니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
```

2. 각 클라이언트에 인증 기관 파일을 복사한 후 각 클라이언트에서 다음 명령을 실행하여 인증 기관 신뢰 번들에 인증서를 추가합니다.

```
$ sudo update-ca-trust extract
```

20.4. SSL/TLS 키 생성

OpenStack 환경에서 **SSL/TLS**를 활성화하려면 인증서를 생성하기 위한 **SSL/TLS** 키가 필요합니다.

절차

1. 다음 명령을 실행하여 **SSL/TLS** 키**(server.key.pem)**를 생성합니다.

```
$ openssl genrsa -out server.key.pem 2048
```

20.5. SSL/TLS 인증서 서명 요청 생성

인증서 서명 요청을 생성하려면 다음 단계를 완료합니다.

절차

1. 기본 **OpenSSL** 설정 파일을 복사합니다.

```
$ cp /etc/pki/tls/openssl.cnf .
```

2. 새 **openssl.cnf** 파일을 편집하고 **director**에 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_default를 다음 항목 중 하나로 설정합니다.

- **IP** 주소를 사용하여 **SSL/TLS**를 통해 **director**에 액세스하는 경우 **undercloud.conf** 파일의 **undercloud_public_host** 매개변수를 사용합니다.
- 정규화된 도메인 이름을 사용하여 **SSL/TLS**을 통해 **director**에 액세스하는 경우 도메인 이름을 사용합니다.

alt_names 섹션을 편집하여 다음 항목을 포함합니다.

- **IP -** 클라이언트가 **SSL**을 통해 **director**에 액세스하는 데 사용하는 **IP** 주소 목록입니다.
- **DNS -** 클라이언트가 **SSL**을 통해 **director**에 액세스하는 데 사용하는 도메인 이름 목록입니다. 또한 공용 **API IP** 주소를 **alt_names** 섹션 끝에 **DNS** 항목으로 추가합니다.

참고

openssl.cnf에 대한 자세한 내용을 보려면 **man openssl.cnf** 명령을 실행합니다.

3. 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 옵션을 사용하여 **OpenStack SSL/TLS** 키를 지정합니다.

이 명령을 실행하면 인증서 서명 요청인 **server.csr.pem** 파일이 생성됩니다. 이 파일을 사용하여 **OpenStack SSL/TLS** 인증서를 생성합니다.

20.6. SSL/TLS 인증서 생성

OpenStack 환경에 대한 **SSL/TLS** 인증서를 생성하려면 다음과 같은 파일이 필요합니다.

openssl.cnf

v3 확장을 지정하는 사용자 지정 설정 파일입니다.

server.csr.pem

인증서를 생성하고 인증 기관을 통해 서명하는 인증서 서명 요청입니다.

ca.crt.pem

인증서에 서명하는 인증 기관입니다.

ca.key.pem

인증 기관 개인 키입니다.

절차

1. 아직 없는 경우 새**certs** 디렉터리를 만듭니다.

```
sudo mkdir -p /etc/pki/CA/newcerts
```

2. 다음 명령을 실행하여 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 다음 옵션을 사용합니다.

-config

사용자 지정 구성 파일, 즉 **v3** 확장이 포함된 **openssl.cnf** 파일을 사용합니다.

-extensions v3_req

v3 확장을 활성화합니다.

-days

인증서가 만료될 때까지 남은 기간(일)을 정의합니다.

-in'

인증서 서명 요청입니다.

-out

생성된 서명된 인증서입니다.

-cert

인증 기관 파일입니다.

-keyfile

인증 기관 개인 키입니다.

이 명령을 실행하면 **server.crt.pem**이라는 새 인증서가 생성됩니다. 이 인증서를 **OpenStack SSL/TLS** 키와 함께 사용합니다.

20.7. 언더클라우드에 인증서 추가

언더클라우드 신뢰 번들에 **OpenStack SSL/TLS** 인증서를 추가하려면 다음 단계를 완료합니다.

절차

1. 다음 명령을 실행하여 인증서와 키를 결합합니다.

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

이 명령을 실행하면 **undercloud.pem** 파일이 생성됩니다.

2. **/etc/pki** 디렉터리 내의 위치에 **undercloud.pem** 파일을 복사하고 **HAProxy**가 읽을 수 있도록 필요한 **SELinux** 컨텍스트를 설정합니다.

```
$ sudo mkdir /etc/pki/undercloud-certs
$ sudo cp ~/undercloud.pem /etc/pki/undercloud-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/undercloud-certs(/.*)?"
$ sudo restorecon -R /etc/pki/undercloud-certs
```

3. **undercloud.conf** 파일의 **undercloud_service_certificate** 옵션에 **undercloud.pem** 파일 위치를 추가합니다.

```
undercloud_service_certificate = /etc/pki/undercloud-certs/undercloud.pem
```

generate_service_certificate 및 **certificate_generation_ca** 매개변수를 설정하거나 활성화하지 마십시오. **director**는 이러한 매개변수를 사용하여 수동으로 생성한 **undercloud.pem** 인증서를 사용하는 대신 인증서를 자동으로 생성합니다.

4. 인증서에 서명한 인증 기관을 언더클라우드의 신뢰할 수 있는 인증 기관 목록에 추가하면 언더클라우드 내의 다른 서비스에서 해당 인증 기관에 액세스할 수 있습니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

언더클라우드에 인증 기관이 추가되었는지 확인하려면 **openssl**을 사용하여 신뢰 번들을 확인합니다.

```
$ openssl crl2pkcs7 -nocrl -certfile /etc/pki/tls/certs/ca-bundle.crt | openssl pkcs7 -print_certs
-text | grep <CN of the CA issuer> -A 10 -B 10
```

- **<CN of the CA issuer>**를 **CA** 발행자의 일반 이름으로 바꿉니다. 이 명령은 유효 날짜를 포함하여 주요 인증서 세부 정보를 출력합니다.

21장. 추가 인트로스펙션 작업

표준 오버클라우드 배포 워크플로우 외부에서 인트로스펙션을 수행해야 하는 경우가 있습니다. 예를 들어, 사용되지 않은 기존 노드에서 하드웨어를 교체한 후 새 노드를 인트로스펙션하거나 인트로스펙션 데이터를 새로 고칠 수 있습니다.

21.1. 개별적으로 노드 인트로스펙션 수행

사용 가능한 노드에서 단일 인트로스펙션을 수행하려면 노드를 관리 모드로 설정하고 인트로스펙션을 수행합니다.

절차

1. 모든 노드를 **manageable** 상태로 설정합니다.

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

2. 인트로스펙션을 수행합니다.

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

인트로스펙션이 완료되면 노드가 **available** 상태로 바뀝니다.

21.2. 초기 인트로스펙션 이후 노드 인트로스펙션 수행

초기 인트로스펙션 이후 모든 노드는 **--provide** 옵션으로 인해 **available** 상태가 됩니다. 초기 인트로스펙션 이후 모든 노드에서 인트로스펙션을 수행하려면 노드를 관리 모드로 설정하고 인트로스펙션을 수행합니다.

절차

1. 모든 노드를 **manageable** 상태로 설정

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

2. 대규모 인트로스펙션 명령을 실행합니다.

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

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

21.3. 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. **director**는 기본적으로 **LLDP** 데이터 수집을 활성화합니다.

절차

1. 노드의 인터페이스 목록을 가져오려면 다음 명령을 실행합니다.

```
(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            |
+-----------+-------------------+------------------------+-------------------+----------------+
```

2. 인터페이스 데이터 및 스위치 포트 정보를 보려면 다음 명령을 실행합니다

```
(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
|
+---------------------------------+-------------------------------------------------------------------------
-----------------------------------------+
```

21.4. 하드웨어 인트로스펙션 세부 정보 검색

기본적으로 **Bare Metal** 서비스 하드웨어 검사 추가 기능이 활성화되므로 이 기능을 사용하여 오버클라우드 구성의 하드웨어 세부 정보를 검색할 수 있습니다. **undercloud.conf** 파일의 **inspection_extras** 매개변수에 대한 자세한 내용은 **director** 구성을 참조하십시오.

예를 들어 **numa_topology** 수집기는 하드웨어 검사 추가 기능에 포함되며 각 **NUMA** 노드에 대해 다음과 같은 정보가 포함되어 있습니다.

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

절차

- 위에 나열된 정보를 검색하려면 **<UUID>**를 베어 메탈 노드의 **UUID**로 바꿔 다음 명령을 완료하십시오.

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

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
    }
  ]
}
```

22장. 베어 메탈 노드 자동 검색

instackenv.json 파일을 생성하지 않고도 자동 검색을 사용하여 오버클라우드 노드를 등록하고 해당 메타데이터를 생성할 수 있습니다. 이 개선 사항으로 초기 노드 정보의 검색 시간을 단축할 수 있습니다. 예를 들어 자동 검색을 사용하면 **IPMI IP** 주소를 조회하고 차후에 **instackenv.json**을 생성할 필요가 없습니다.

22.1. 자동 검색 활성화

PXE로 부팅할 때 프로비저닝 네트워크에 연결된 노드를 자동으로 검색하고 가져오도록 베어 메탈 자동 검색을 활성화하고 구성합니다.

절차

1. **undercloud.conf** 파일에서 **Bare Metal** 자동 검색을 활성화합니다.

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

- **enable_node_discovery -** 활성화하면 **PXE**를 사용하여 인트로스펙션 램디스크를 부팅하는 모든 노드가 **Bare Metal** 서비스(**ironic**)에 자동으로 등록됩니다.
- **discovery_default_driver -** 검색된 노드에 사용할 드라이버를 설정합니다. 예를 들어 **ipmi**입니다.

2. **ironic**에 **IPMI** 인증서를 추가합니다.

a. **ipmi-credentials.json**이라는 파일에 **IPMI** 인증서를 추가합니다. 환경에 맞게 이 예제의 **SampleUsername, RedactedSecurePassword** 및 **bmc_address** 값을 변경합니다.

```
[
  {
    "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
```

22.2. 자동 검색 테스트

베어 메탈 자동 검색 기능을 테스트하기 위해 프로비저닝 네트워크에 연결된 노드를 PXE 부팅합니다.

절차

1. **필요한 노드의 전원을 켭니다.**

2. **openstack baremetal node list 명령을 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
```

22.3. 규칙을 사용하여 다른 벤더 하드웨어 검색

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

절차

1. 다음 콘텐츠로 **dell-drac-rules.json**이라는 파일을 생성합니다.

```
[
  {
    "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]}"}
       ]
   }
]
```

- 이 예제의 사용자 이름 및 암호 값을 사용자 환경에 맞게 변경합니다.

2. 규칙을 **ironic**으로 가져옵니다.

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

23장. 자동 프로필 태그 설정

인트로스펙션 프로세스는 일련의 벤치마크 테스트를 수행합니다. director는 이러한 테스트의 데이터를 저장합니다. 이 데이터를 사용하는 정책 세트를 다양한 방법으로 생성할 수 있습니다.

- 정책을 통해 성능이 떨어지거나 불안정한 노드를 식별하고 오버클라우드에서 사용되지 않도록 분리할 수 있습니다.
- 정책을 통해 노드를 특정 프로필에 자동으로 태그할지 여부를 정의할 수 있습니다.

23.1. 정책 파일 구문

정책 파일은 규칙 세트가 포함된 JSON 포멧을 사용합니다. 각 규칙은 설명, 조건 및 작업을 정의합니다. 설명은 규칙의 일반 텍스트 설명이고, 조건은 키-값 패턴을 사용하여 평가를 정의하며, 작업은 조건의 수행입니다.

설명

설명은 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.

- memory_mb - 노드의 메모리 크기(MB)
- cpus - 노드 CPU의 총 스레드 수

- **cpu_arch -** 노드 **CPU**의 아키텍처
- **local_gb -** 노드 **root** 디스크의 총 스토리지 공간

op

평가에 사용할 작업을 정의합니다. 다음과 같은 속성이 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 정규식과 완전히 일치해야 함
- **contains -** 지정된 정규식을 포함하는 값이 필요함
- **is-empty -** 필드가 비어 있는지 확인

invert

평가 결과를 반전할지 여부를 정의하는 부울 값입니다.

multiple

여러 결과가 있는 경우 사용할 평가를 정의합니다. 이 매개변수에는 다음과 같은 속성이 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. 값이 아니면 조건이 **false** 결과를 반환합니다.

예제:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

작업

조건이 **true**이면 정책이 작업을 수행합니다. 작업은 **action** 값에 따라 **action** 키와 추가 키를 사용합니다.

- **fail -** 인트로스펙션이 실패합니다. 실패 메시지에 대한 **message** 매개변수가 필요합니다.
- **set-attribute - Ironic** 노드의 속성을 설정합니다. **Ironic** 속성의 경로(예: **/driver_info/ipmi_address**)인 **path** 필드와 설정할 **value**가 필요합니다.

- **set-capability - ironic** 노드의 기능을 설정합니다. 새 기능의 이름과 값인 **name** 및 **value** 필드가 필요합니다. 이 기능의 기존 값을 대체합니다. 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 **True**로 설정하면, 지정된 값이 이미 목록에 있을 경우 아무것도 추가되지 않습니다.

예제:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

23.2. 정책 파일 예제

다음은 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. 클라우드에서 특정 노드를 제외하려는 경우 이러한 유형의 규칙을 적용할 수 있습니다.**
- **하드 드라이브 크기가 1TiB 이상인 노드에는 무조건 swift-storage 프로필이 할당됩니다.**

- 하드 드라이브 크기가 **40GiB**보다 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 할당할 수 있습니다. 이 프로세스가 성공하려면 기존 프로필 기능을 삭제해야 합니다. 삭제하지 않으면 기존 프로필 기능이 우선합니다.

프로필 일치 규칙은 다른 노드를 변경하지 않습니다.

참고

인트로스펙션 규칙을 사용하여 **profile** 기능을 할당하면 기존 값이 항상 재정의됩니다. 하지만 이미 프로필 기능이 있는 노드에 대해서는 **[PROFILE]_profile** 기능이 무시됩니다.

23.3. 정책 파일 가져오기

정책 파일을 **director**로 가져오려면 다음 단계를 완료합니다.

절차

1. 정책 파일을 **director**로 가져옵니다.

```
$ openstack baremetal introspection rule import rules.json
```

2. 인트로스펙션 프로세스를 실행합니다.

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

3. 인트로스펙션이 완료되면 노드 및 할당된 해당 프로필을 확인합니다.

```
$ openstack overcloud profiles list
```

4. 인트로스펙션 규칙에 오류가 있을 경우 다음 명령을 실행하여 모든 규칙을 삭제합니다.

```
$ openstack baremetal introspection rule purge
```

24장. 전체 디스크 이미지 생성

기본 오버클라우드 이미지는 파티셔닝 정보 또는 부트로더가 포함되지 않은 플랫 파티션 이미지입니다. **director**는 노드를 부팅할 때 별도의 커널 및 램디스크를 사용하고 오버클라우드 이미지를 디스크에 쓸 때 기본 파티션 레이아웃을 생성합니다. 그러나 파티션 레이아웃, 부트로더 및 강화된 보안 기능이 포함된 전체 디스크 이미지를 생성할 수 있습니다.

중요

다음 프로세스에서는 **director** 이미지 빌드 기능을 사용합니다. **Red Hat**은 이 섹션에 포함된 지침을 사용하는 이미지만 지원합니다. 이 사양과 다르게 빌드된 사용자 지정 이미지는 지원되지 않습니다.

24.1. 보안 강화 조치

전체 디스크 이미지에는 보안이 중요한 기능인 **Red Hat OpenStack Platform** 배포에 필요한 추가 보안 강화 조치가 포함되어 있습니다.

이미지 생성을 위한 보안 권장 사항

- **/tmp** 디렉터리가 별도의 볼륨이나 파티션에 마운트되어 있고 **rw, nosuid, nodev, noexec, relatime** 플래그가 있습니다.
- **/var, /var/log, /var/log/audit** 디렉터리는 별도의 볼륨이나 파티션에 마운트되어 있고 **rw** 및 **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** 커널 부트 플래그를 추가합니다.
  - 부트로더 구성을 사용하는 **USB**에 대해 커널 지원을 비활성화하려면 **nousb**를 추가합니다.

- 비보안 부트 플래그를 삭제하려면 **crashkernel=auto**를 제거합니다.

- 비보안 모듈(**usb-storage, cramfs, freevxfs, jffs2, hfs, hfsplus, squashfs, udf, vfat**)을 블랙리스트로 지정하고 이러한 모듈이 로드되지 않도록 합니다.

- 기본적으로 설치되므로 이미지에서 비보안 패키지(**kexec-tools** 및 **telnet**이 설치한 **kdump**)를 삭제합니다.

24.2. 전체 디스크 이미지 워크플로우

전체 디스크 이미지를 빌드하려면 다음 워크플로우를 완료합니다.

1. 기본 **Red Hat Enterprise Linux 8.4** 이미지를 다운로드합니다.
2. 등록과 관련된 환경 변수를 설정합니다.
3. 파티션 스키마 및 크기를 수정하여 이미지를 사용자 지정합니다.
4. 이미지를 생성합니다.
5. **director**에 이미지를 업로드합니다.

24.3. 기본 클라우드 이미지 다운로드

전체 디스크 이미지를 빌드하기 전에 기준으로 사용할 **Red Hat Enterprise Linux**의 기존 클라우드 이미지를 다운로드해야 합니다.

절차

1. **Red Hat Enterprise Linux 8.4** 다운로드 페이지로 이동합니다. **Red Hat OpenStack Platform 16.2**는 **Red Hat Enterprise Linux 8.4**에서 지원됩니다.

https://access.redhat.com/downloads/content/479/ver=/rhel---8/8.4/x86_64/product-software

참고

프롬프트가 표시되면 고객 포털 로그인 정보를 입력합니다.

2. **Red Hat Enterprise Linux 8.4 KVM** 게스트 이미지 옆에 있는 지금 다운로드를 클릭합니다.

24.4. 일관된 인터페이스 이름 지정 활성화

기본적으로 **KVM** 게스트 이미지에서 일관된 네트워크 인터페이스 장치 이름 지정이 비활성화됩니다. **virt-customize** 를 사용하여 일관된 이름을 활성화합니다.

절차

1. **KVM** 게스트 이미지를 **/var/lib/libvirt/images** 로 이동합니다.

```
$ sudo mv <kvm_guest_image> /var/lib/libvirt/images/
```

2. **libvirtd** 를 시작합니다.

```
$ sudo systemctl start libvirtd
```

3. **KVM** 게스트 이미지에서 일관된 인터페이스 이름 지정을 활성화합니다.

```
$ sudo virt-customize -a /var/lib/libvirt/images/<kvm guest image> --edit
/etc/default/grub:s/net.ifnames=0/net.ifnames=1/
```

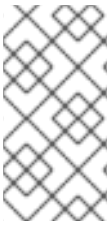
4. **libvirtd** 중지:

```
$ sudo systemctl stop libvirtd
```

24.5. 디스크 이미지 환경 변수

디스크 이미지 빌드 프로세스의 일환으로 **director**는 새 오버클라우드 이미지 패키지를 가져오기 위해

기본 이미지와 등록 세부 정보가 필요합니다. 다음 **Linux** 환경 변수를 사용하여 이러한 속성을 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

전체 디스크 이미지의 기반으로 사용할 로컬 이미지를 설정합니다.

REG_ACTIVATION_KEY

로그인 정보 대신 활성화 키를 등록 프로세스의 일부로 사용합니다.

REG_AUTO_ATTACH

가장 호환되는 서브스크립션을 자동으로 연결할지 여부를 정의합니다.

REG_BASE_URL

이미지용 패키지가 포함된 콘텐츠 전달 서버의 기본 **URL**입니다. 기본 **Customer Portal** 서브스크립션 관리 프로세스는 **https://cdn.redhat.com**을 사용합니다. **Red Hat Satellite 6** 서버를 사용하는 경우 이 매개변수를 **Satellite** 서버의 기본 **URL**로 설정합니다.

REG_ENVIRONMENT

조직 내의 환경에 등록합니다.

REG_METHOD

등록 방법을 설정합니다. **Red Hat** 고객 포털에 시스템을 등록하려면 **portal**을 사용합니다. **Red Hat Satellite 6**에 시스템을 등록하려면 **satellite**를 사용합니다.

REG_ORG

이미지를 등록할 조직입니다.

REG_POOL_ID

제품 서브스크립션 정보의 풀 **ID**입니다.

REG_PASSWORD

이미지를 등록하는 사용자 계정의 암호를 설정합니다.

REG_RELEASE

Red Hat Enterprise Linux 마이너 릴리스 버전을 설정합니다. **REG_AUTO_ATTACH** 또는 **POOL_ID** 환경 변수와 함께 사용해야 합니다.

REG_REPOS

리포지토리 이름의 쉼표로 구분된 문자열입니다. 이 문자열의 각 리포지토리는 **subscription-manager**를 통해 활성화됩니다.

보안이 강화된 전체 디스크 이미지에 다음 리포지토리를 사용합니다.

- **rhel-8-for-x86_64-baseos-eus-rpms**
- **rhel-8-for-x86_64-appstream-eus-rpms**
- **rhel-8-for-x86_64-highavailability-eus-rpms**
- **ansible-2.9-for-rhel-8-x86_64-rpms**
- **fast-datapath-for-rhel-8-x86_64-rpms**
- **openstack-16.2-for-rhel-8-x86_64-rpms**

REG_SAT_URL

오버클라우드 노드를 등록할 **Satellite** 서버의 기본 **URL**입니다. **Satellite HTTPS URL** 대신 **HTTP URL**을 이 매개변수에 사용합니다. 예를 들어 **https://satellite.example.com** 대신 **http://satellite.example.com**을 사용합니다.

REG_SERVER_URL

사용할 서브스크립션 서비스의 호스트 이름을 설정합니다. **Red Hat Customer Portal**의 기본 호스트 이름은 **subscription.rhn.redhat.com**에 있습니다. **Red Hat Satellite 6** 서버를 사용하는 경우 이

매개변수를 **Satellite** 서버의 호스트 이름으로 설정하십시오.

REG_USER

이미지를 등록할 계정의 사용자 이름을 설정합니다.

환경 변수 세트를 내보내고 로컬 **QCOW2** 이미지를 **Red Hat** 고객 포털에 임시로 등록하려면 다음 예제 명령 세트를 사용합니다.

```
$ export DIB_LOCAL_IMAGE=./rhel-8.4-x86_64-kvm.qcow2
$ export REG_METHOD=portal
$ export REG_USER=<your_name>
$ export REG_PASSWORD=<your_password>
$ export REG_RELEASE="8.4"
$ export REG_POOL_ID=<pool_id>
$ export REG_REPOS="rhel-8-for-x86_64-baseos-eus-rpms \
    rhel-8-for-x86_64-appstream-eus-rpms \
    rhel-8-for-x86_64-highavailability-eus-rpms \
    ansible-2.9-for-rhel-8-x86_64-rpms \
    fast-datapath-for-rhel-8-x86_64-rpms \
    openstack-16.2-for-rhel-8-x86_64-rpms"
```

24.6. 디스크 레이아웃 사용자 지정

기본 보안 강화 이미지 크기는 **20G**이며 사전 정의된 파티션 크기를 사용합니다. 하지만 오버클라우드 컨테이너 이미지를 수용하려면 파티션 레이아웃을 수정해야 합니다. 이미지 크기를 **40G**로 늘리려면 다음 섹션의 단계를 완료합니다. 요구에 맞게 파티션 레이아웃과 디스크 크기를 추가로 수정할 수 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.

24.7. 파티션 스키마 수정

파티션 스키마를 수정하여 파티션 크기를 변경하거나, 새 파티션을 생성하거나, 기존 파티션을 삭제할 수 있습니다. 다음 환경 변수를 사용하여 새 파티션 스키마를 정의합니다.

```
$ export DIB_BLOCK_DEVICE_CONFIG='<yaml_schema_with_partitions>'
```

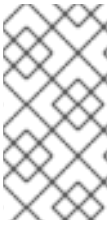
BIOS 예

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

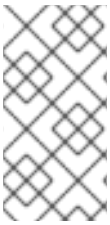
배포 후에는 파티션 크기를 조정할 수 없으므로 이미지에 대해 올바른 파티션 크기를 정의해야 합니다.

UEFI 예

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: gpt
    partitions:
      - name: ESP
        type: 'EF00'
        size: 200MiB
      - name: BSP
        type: 'EF02'
        size: 1MiB
      - name: ROOT
        type: '8300'
        size: 100%
- mkfs:
    name: fs_esp
    base: ESP
    type: vfat
    mount:
      mount_point: /boot/efi
      fstab:
        options: "defaults"
        fsck-passno: 1
- mkfs:
    name: fs_root
    label: "img-rootfs"
    base: ROOT
    type: xfs
    mount:
      mount_point: /
      fstab:
        options: "defaults"
        fsck-passno: 1
'''
```

이 샘플 **YAML** 콘텐츠를 이미지 파티션 스키마의 기준으로 사용합니다. 환경에 맞게 파티션 크기와 레이아웃을 수정합니다.



참고

배포 후에는 파티션 크기를 조정할 수 없으므로 배포 전에 이미지에 대해 올바른 파티션 크기를 정의해야 합니다.

24.8. 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 설정 파일을 편집합니다.

절차

1. **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml** 복사본을 생성합니다.

```
# cp /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml \
/home/stack/overcloud-hardened-images-python3-custom.yaml
```

참고

UEFI 전체 디스크 이미지에 **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-uefi-python3.yaml**을 사용합니다.

2. 설정 파일의 **DIB_IMAGE_SIZE**를 편집하고 필요에 따라 값을 조정합니다.

```
...

environment:
  DIB_PYTHON_VERSION: '3'
  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.

3. 선택 사항: 프록시를 구성하려면 **http_proxy** 및 **https:_proxy** 환경 변수도 포함해야 합니다.

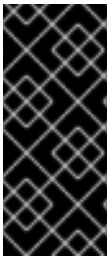
```
environment:
  http_proxy: <proxy_server>
  https_proxy: <proxy_server>
```

- **& lt;proxy_server >**를 프록시 주소로 바꿉니다.

4. 파일을 저장합니다.

중요

오버클라우드를 배포하면 **director**는 오버클라우드 이미지의 **RAW** 버전을 생성합니다. 따라서 언더클라우드에 **RAW** 이미지를 수용하기에 충분한 여유 공간이 있어야 합니다. 예를 들어 보안 강화 이미지 크기를 **40G**로 설정하는 경우 언더클라우드 하드 디스크에 **40G**의 사용 가능한 공간이 있어야 합니다.



중요

director는 물리 디스크에 이미지를 쓸 때 디스크 끝에 **64MB**의 구성 드라이브 주 파티션을 생성합니다. 전체 디스크 이미지를 생성하는 경우 물리 디스크의 크기가 이 추가 파티션을 수용하는지 확인합니다.

24.9. 전체 디스크 이미지 빌드

환경 변수를 설정하고 이미지를 사용자 지정한 후 **openstack overcloud image build** 명령을 사용하여 이미지를 생성합니다.

절차

1. 필요한 모든 설정 파일을 사용하여 **openstack overcloud image build** 명령을 실행합니다.

```
# openstack overcloud image build \
--image-name overcloud-hardened-full \ 1
--config-file /home/stack/overcloud-hardened-images-python3-custom.yaml \ 2
--config-file /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-rhel8.yaml 3
```

1

UEFI 전체 디스크 이미지의 경우 **overcloud-hardened-uefi-full** 을 사용합니다.

2

overcloud-hardened-images-python3-custom.yaml 파일은 새 디스크 크기가 포함된 사용자 지정 설정 파일입니다. 다른 사용자 지정 디스크 크기를 사용하지 않는 경우 기존 **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml** 파일을 대신 사용합니다. 표준 **UEFI** 전체 디스크 이미지의 경우 **overcloud-hardened-images-uefi-python3.yaml**을 사용합니다.

3

UEFI 전체 디스크 이미지의 경우 **overcloud-hardened-images-uefi-rhel8.yaml**을 사용합니다.

이 명령을 실행하면 **overcloud-hardened-full.qcow2**라는 이미지가 생성되고 필요한 모든 보안 기능이 포함됩니다.

24.10. 전체 디스크 이미지 업로드

이미지를 **OpenStack Image(glance)** 서비스에 업로드하고 **Red Hat OpenStack Platform director**에서 이를 사용하기 시작합니다. 보안 강화 이미지를 업로드하려면 다음 단계를 완료합니다.

1. 새로 생성된 이미지의 이름을 변경하고 이미지를 **images** 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
```

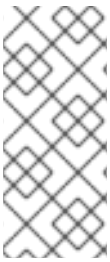
3. 새 오버클라우드 이미지를 업로드합니다.

```
# openstack overcloud image upload --image-path /home/stack/images --whole-disk
```

기존 이미지를 보안 강화 이미지로 교체하려면 **--update-existing** 플래그를 사용합니다. 이 플래그는 기존 **overcloud-full** 이미지를 보안 강화된 새로운 이미지로 덮어씁니다.

25장. 직접 배포 설정

노드를 프로비저닝할 때 **director**는 오버클라우드 기본 운영 체제 이미지를 **iSCSI** 마운트에 마운트한 다음, 각 노드의 디스크에 이미지를 복사합니다. 직접 배포는 **HTTP** 위치의 디스크 이미지를 베어 메탈 노드의 디스크에 직접 쓰는 대체 방법입니다.

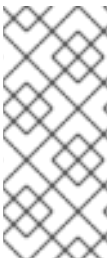


참고

iSCSI 배포 인터페이스 **iscsi** 는 **RHOSP(Red Hat OpenStack Platform)** 버전 **17.0**에서 더 이상 사용되지 않으며 **RHOSP 18.0**에서 제거됩니다. 직접 배포(**Direct**)는 **RHOSP 17.0**의 기본 배포 인터페이스입니다.

25.1. 언더클라우드에 직접 배포 인터페이스 설정

iSCSI 배포 인터페이스가 기본 배포 인터페이스입니다. 그러나 직접 배포 인터페이스를 사용하여 **HTTP** 위치에서 대상 디스크로 이미지를 다운로드할 수 있습니다.



참고

iSCSI 배포 인터페이스에 대한 지원은 **RHOSP(Red Hat OpenStack Platform)** 버전 **17.0**에서 더 이상 사용되지 않으며 **RHOSP 18.0**에서 제거됩니다. **RHOSP 17.0**의 기본 배포 인터페이스가 직접 배포됩니다.

사전 요구 사항

- 오버클라우드 노드 메모리 **tmpfs**에는 **8GB** 이상의 **RAM**이 있어야 합니다.

절차

1. 사용자 지정 환경 파일 **/home/stack/undercloud_custom_env.yaml**을 생성하거나 수정하고 **IronicDefaultDeployInterface**를 지정합니다.

```
parameter_defaults:
  IronicDefaultDeployInterface: direct
```

2. 기본적으로 각 노드의 **Bare Metal** 서비스(**ironic**) 에이전트는 **HTTP** 링크를 통해 **Object Storage** 서비스(**swift**)에 저장된 이미지를 가져옵니다. 또는 **ironic**은 **ironic-conductor HTTP** 서버를 통해 이 이미지를 노드로 직접 스트리밍할 수 있습니다. 이미지 제공 서비스를 변경하려면

/home/stack/undercloud_custom_env.yaml 파일에서 **IronicImageDownloadSource**를 **http**로 설정합니다.

```
parameter_defaults:
  IronicDefaultDeployInterface: direct
  IronicImageDownloadSource: http
```

3. **undercloud.conf** 파일의 **DEFAULT** 섹션에 사용자 지정 환경 파일을 추가합니다.

```
custom_env_files = /home/stack/undercloud_custom_env.yaml
```

4. 언더클라우드 설치를 수행합니다.

```
$ openstack undercloud install
```

26장. 가상화된 컨트롤 플레인 생성

가상화된 컨트롤 플레인은 베어 메탈이 아닌 VM(가상 머신)에 있는 컨트롤 플레인입니다. 가상화된 컨트롤 플레인을 사용하여 컨트롤 플레인에 필요한 베어 메탈 머신 수를 줄입니다.

이 장에서는 RHOSP(Red Hat OpenStack Platform) 및 Red Hat Virtualization을 사용하여 RHOSP 컨트롤 플레인을 가상화하는 방법을 설명합니다.

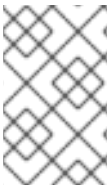
26.1. 가상화된 컨트롤 플레인 아키텍처

director를 통해 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.

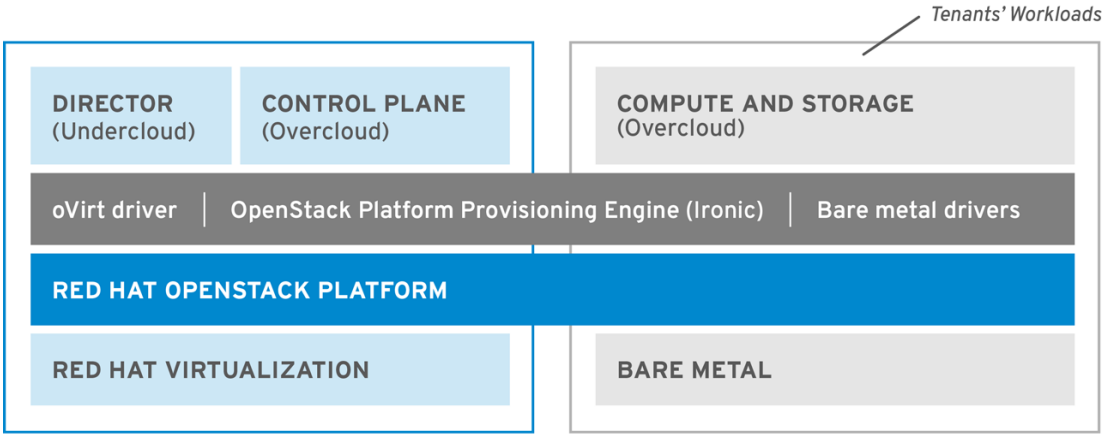
다음 아키텍처 다이어그램에서는 가상화된 컨트롤 플레인 배포 방법을 설명합니다. Red Hat Virtualization의 VM에서 실행 중인 컨트롤러 노드로 오버클라우드를 배포하고 베어 메탈에서 컴퓨팅 및 스토리지 노드를 실행합니다.



참고

Red Hat Virtualization에서 OpenStack 가상화된 언더클라우드를 실행합니다.

가상화된 컨트롤 플레인 아키텍처



OpenStack Bare Metal Provisioning 서비스(ironic)에는 **Red Hat Virtualization VM** 드라이버 **staging-ovirt**가 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.

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 다운타임을 방지하고 성능을 향상시킬 수 있는 다양한 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 **VM**을 배포할 수 있습니다. 이는 데이터 센터의 서버 공간 사용을 최소화하고 물리적 노드의 효율성을 극대화합니다.
- 구성 가능 역할을 사용하여 보다 복잡한 **RHOSP** 컨트롤 플레인을 정의하고, 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. cinder-volume 전용 역할을 생성하고 이 역할을 사용하여 가상화된 컨트롤러에 이를 포함하는 대신 물리 노드를 생성해야 합니다. 자세한 내용은 Composable Services and Custom Roles를 참조하십시오.

26.2. RED HAT VIRTUALIZATION 드라이버를 사용하여 가상화된 컨트롤러 프로비저닝

RHOSP 및 Red Hat Virtualization을 사용하여 오버클라우드의 가상화된 RHOSP 컨트롤 플레인을 프로비저닝하려면 다음 단계를 완료합니다.

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 Controller Node Requirements를 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, 호스트 네트워크의 클러스터가 여러 네트워크에서 네트워크 분리를 사용할 준비가 되어 있어야 합니다. 자세한 내용은 Logical Networks를 참조하십시오.

- 시간대 오프셋을 적용하기 전에 hwclock이 BIOS 클럭을 동기화할 때 타임 스탬프에 미래 날짜가 설정되는 문제를 방지하려면 각 노드의 내부 BIOS 클럭을 UTC로 설정해야 합니다.

작은 정보

성능 병목 현상을 방지하려면 구성 가능 역할을 사용하고 베어 메탈 컨트롤러 노드에 데이터 플레인 서비스를 유지합니다.

절차

1. director에서 staging-ovirt 드라이버를 활성화하려면 undercloud.conf 구성 파일의 enabled_hardware_types에 해당 드라이버를 추가합니다.

```
enabled_hardware_types = ipmi,redfish,ilo,idrac,staging-ovirt
```

2. 언더클라우드에 staging-ovirt 드라이버가 있는지 확인합니다.

```
(undercloud) [stack@undercloud ~]$ openstack baremetal driver list
```

언더클라우드를 올바르게 설정한 경우 이 명령은 다음과 같은 결과를 반환합니다.

```
+---------------------+----------------------+
| 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. 오버클라우드 노드 정의 템플릿(예: **nodes.json**)을 업데이트하여 **Red Hat Virtualization**에 호스트된 **VM**을 **director**에 등록합니다. 자세한 내용은 오버클라우드 노드 등록을 참조하십시오. 다음 키:값 쌍을 사용하여 오버클라우드와 함께 배포할 **VM**의 속성을 정의합니다.

표 26.1. 오버클라우드에 대한 VM 설정

키	이 값으로 설정
pm_type	oVirt/RHV VM용 OpenStack Bare Metal Provisioning(ironic) 서비스 드라이버인 **staging-ovirt**
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":"{osp_curr_ver}-controller-0",
            "capabilities": "profile:control,boot_option:local"
        },
        ...
}
```

각 **Red Hat Virtualization** 호스트에서 하나의 컨트롤러 구성

4. Red Hat Virtualization에서 "soft negative affinity"로 선호도 그룹을 구성하여 컨트롤러 VM에 대해 고가용성이 구현되었는지 확인합니다. 자세한 내용은 Affinity Groups를 참조하십시오.

5. Red Hat Virtualization Manager 인터페이스를 열고 이를 사용하여 각 VLAN을 컨트롤러 VM의 개별 논리 vNIC에 매핑합니다. 자세한 내용은 Logical Networks를 참조하십시오.

6. director와 컨트롤러 VM의 vNIC에서 no_filter를 설정하고 VM을 다시 시작하여 컨트롤러 VM에 연결된 네트워크에서 MAC 스푸핑 필터를 비활성화합니다. 자세한 내용은 Virtual Network Interface Cards를 참조하십시오.

7. 오버클라우드를 배포하여 새 가상 컨트롤러 노드를 환경에 추가합니다.

```
(undercloud) [stack@undercloud ~]$ openstack overcloud deploy --templates
```

27장. 고급 컨테이너 이미지 관리 수행

기본 컨테이너 이미지 구성은 대부분의 환경에 적합합니다. 경우에 따라 컨테이너 이미지 구성에 버전 고정과 같은 사용자 지정이 필요할 수 있습니다.

27.1. 언더클라우드에서 컨테이너 이미지 고정

특정 상황에서 언더클라우드에 특정 컨테이너 이미지 버전 집합이 필요할 수 있습니다. 이 경우 이미지를 특정 버전에 고정해야 합니다. 이미지를 고정하려면 컨테이너 구성 파일을 생성 및 수정한 다음 언더클라우드 역할 데이터를 컨테이너 구성 파일과 결합하여 서비스 매핑이 포함된 환경 파일을 컨테이너 이미지로 생성해야 합니다. 그런 다음 **undercloud.conf** 파일의 **custom_env_files** 매개변수에 이 환경 파일을 포함합니다.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **--output-env-file** 옵션과 함께 **openstack tripleo container image prepare default** 명령을 실행하여 기본 이미지 구성이 포함된 파일을 생성합니다.

```
$ sudo openstack tripleo container image prepare default \
--output-env-file undercloud-container-image-prepare.yaml
```

3. 환경 요구 사항에 따라 **undercloud-container-image-prepare.yaml** 파일을 수정합니다.

   a. **director**에서 **tag_from_label:** 매개변수를 사용할 수 있도록 **tag:** 매개변수를 제거합니다. **director**는 이 매개변수를 사용하여 각 컨테이너 이미지의 최신 버전을 식별하고, 각 이미지를 가져오고, **director**의 컨테이너 레지스트리에 있는 각 이미지에 태그를 지정합니다.

   b. 언더클라우드의 **Ceph** 레이블을 제거합니다.

   c. **neutron_driver:** 매개변수가 비어 있는지 확인합니다. **OVN**은 언더클라우드에서 지원되지 않으므로 이 매개변수를 **OVN**으로 설정하지 마십시오.

   d. 컨테이너 이미지 레지스트리 인증 정보를 포함합니다.

```
ContainerImageRegistryCredentials:
  registry.redhat.io
    myser: 'p@55w0rd!'
```

참고

image-serve 레지스트리가 아직 설치되지 않았으므로 컨테이너 이미지를 새 언더클라우드의 언더클라우드 레지스트리로 내보낼 수 없습니다. **push_destination** 값을 **false**로 설정하거나 사용자 지정 값을 사용하여 소스에서 직접 이미지를 가져와야 합니다. 자세한 내용은 컨테이너 이미지 준비 매개변수를 참조하십시오.

4. 사용자 지정 **undercloud-container-image-prepare.yaml** 파일과 결합된 언더클라우드 역할 파일을 사용하는 새 컨테이너 이미지 구성 파일을 생성합니다.

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

undercloud-container-images.yaml 파일은 서비스 매개변수와 컨테이너 이미지 매핑이 포함된 환경 파일입니다. 예를 들어 **OpenStack Identity(keystone)**는 **ContainerKeystoneImage** 매개변수를 사용하여 컨테이너 이미지를 정의합니다.

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.2.4-5
```

컨테이너 이미지 태그는 **{version}-{release}** 포맷과 일치합니다.

5. **undercloud.conf** 파일의 **custom_env_files** 매개변수에 **undercloud-container-images.yaml** 파일을 포함합니다. 언더클라우드 설치를 실행하면 언더클라우드 서비스에서 이 파일에서 고정된 컨테이너 이미지 매핑을 사용합니다.

27.2. 오버클라우드의 컨테이너 이미지 고정

특정 상황에서는 오버클라우드에 특정 컨테이너 이미지 버전 세트가 필요할 수 있습니다. 이 경우 이미지를 특정 버전에 고정해야 합니다. 이미지를 고정하려면 **containers-prepare-parameter.yaml** 파일을 생성하고 이 파일을 사용하여 컨테이너 이미지를 언더클라우드 레지스트리로 가져오고 고정된 이미지 목록이 포함된 환경 파일을 생성해야 합니다.

예를 들어 **containers-prepare-parameter.yaml** 파일에 다음 내용이 포함될 수 있습니다.

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        name_prefix: openstack-
        name_suffix: ''
        namespace: registry.redhat.io/rhosp-rhel8
        neutron_driver: ovn
      tag_from_label: '{version}-{release}'

  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
```

ContainerImagePrepare 매개변수에는 단일 규칙 **set**가 포함되어 있습니다. 이 규칙 **set**에는 **tag** 매개변수가 포함되지 않아야 하며 **tag_from_label** 매개변수를 사용하여 각 컨테이너 이미지의 최신 버전 및 릴리스를 식별해야 합니다. **director**는 이 규칙 **set**를 사용하여 각 컨테이너 이미지의 최신 버전을 식별하고, 각 이미지를 가져오고, **director**의 컨테이너 레지스트리에 각 이미지에 태그를 지정합니다.

절차

1. **openstack tripleo container image prepare** 명령을 실행하여 **containers-prepare-parameter.yaml** 파일에 정의된 소스에서 모든 이미지를 가져옵니다. **--output-env-file**을 포함하여 고정된 컨테이너 이미지 목록을 포함할 출력 파일을 지정합니다.

```
$ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
```

overcloud-images.yaml 파일은 서비스 매개변수와 컨테이너 이미지 매핑이 포함된 환경 파일입니다. 예를 들어 **OpenStack Identity(keystone)**는 **ContainerKeystoneImage** 매개변수를 사용하여 컨테이너 이미지를 정의합니다.

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.2.4-5
```

컨테이너 이미지 태그는 **{version}-{release}** 포맷과 일치합니다.

2. **openstack overcloud deploy** 명령을 실행할 때 환경 파일 컬렉션을 사용하여 특정 순서로 **containers-prepare-parameter.yaml** 및 **overcloud-images.yaml** 파일을 포함합니다.

```
$ openstack overcloud deploy --templates \
```

```
...
-e /home/stack/containers-prepare-parameter.yaml \
-e /home/stack/overcloud-images.yaml \
...
```

오버클라우드 서비스에서는 **overcloud-images.yaml** 파일에 나열된 고정 이미지를 사용합니다.

28장. DIRECTOR 오류 문제 해결

director 프로세스의 특정 단계에서 오류가 발생할 수 있습니다. 이 섹션에서는 일반적인 문제 진단 방법에 대해 설명합니다.

28.1. 노드 등록 문제 해결

노드 등록 관련 문제는 일반적으로 잘못된 노드 세부 정보 문제로 인해 발생합니다. 이 경우에는 노드 세부 정보가 포함된 템플릿 파일을 확인하고 가져온 노드 세부 정보를 올바르게 수정합니다.

절차

1. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. --validate-only 옵션과 함께 node import 명령을 실행합니다. 이 옵션을 사용하면 가져오기를 수행하지 않고 노드 템플릿이 확인됩니다.

```
(undercloud) $ openstack overcloud node import --validate-only ~/nodes.json
Waiting for messages on queue 'tripleo' with no timeout.

Successfully validated environment file
```

3. 가져온 노드에서 잘못된 세부 정보를 수정하려면 openstack baremetal 명령을 실행하여 노드 세부 정보를 업데이트합니다. 다음 예제에서는 네트워킹 세부 정보를 변경하는 방법을 보여줍니다.

   a. 가져온 노드에 할당된 포트 UUID를 식별합니다.

```
$ source ~/stackrc
(undercloud) $ openstack baremetal port list --node [NODE UUID]
```

   b. MAC 주소를 업데이트합니다.

```
(undercloud) $ openstack baremetal port set --address=[NEW MAC] [PORT UUID]
```

c. 노드에 새 **IPMI** 주소를 설정합니다.

```
(undercloud) $ openstack baremetal node set --driver-info ipmi_address=[NEW IPMI ADDRESS] [NODE UUID]
```

28.2. 하드웨어 인트로스펙션 문제 해결

인트로스펙션 프로세스 실행을 완료해야 합니다. 하지만 검사 램디스크가 응답하지 않을 경우 **ironic-inspector**가 **1**시간(기본값) 후에 시간 초과됩니다. 검사 램디스크의 버그가 원인인 경우도 있지만, 일반적으로는 이 시간 초과는 특히 **BIOS** 부팅 설정 등의 잘못된 환경 구성으로 인해 발생합니다.

일반적인 환경 설정 오류를 진단하고 해결하려면 다음 단계를 수행합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. **director**는 **OpenStack Object Storage(swift)**를 사용하여 인트로스펙션 프로세스 중에 가져오는 하드웨어 데이터를 저장합니다. 이 서비스가 실행 중이 아니면 인트로스펙션이 실패할 수 있습니다. **OpenStack Object Storage**와 관련된 모든 서비스를 확인하여 서비스가 실행 중인지 확인합니다.

```
(undercloud) $ sudo systemctl list-units tripleo_swift*
```

3. 노드가 **manageable** 상태인지 확인합니다. 인트로스펙션은 배포가 가능함을 나타내는 **available** 상태의 노드를 검사하지 않습니다. **available** 상태인 노드를 검사하려면 인트로스펙션 전에 노드 상태를 **manageable** 상태로 변경합니다.

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

4. 인트로스펙션 램디스크에 대한 임시 액세스를 설정합니다. 인트로스펙션 디버깅 중에 노드에 액세스하는 데 사용할 임시 암호 또는 **SSH** 키를 제공할 수 있습니다. 램디스크 액세스를 구성하려면 다음 절차를 완료합니다.

a. 임시 암호로 **openssl passwd -1** 명령을 실행하여 **MD5** 해시를 생성합니다.

```
(undercloud) $ openssl passwd -1 mytestpassword
$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
```

b. **/var/lib/ironic/httpboot/inspector.ipxe** 파일을 편집하고, **kernel**로 시작하는 행을 찾은 다음, **rootpwd** 매개변수 및 **MD5** 해시를 추가합니다.

```
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 **sshkey** 매개변수에 공용 **SSH** 키를 추가합니다.

참고

rootpwd와 **sshkey** 매개변수에 모두 따옴표를 포함합니다.

5. 노드에서 인트로스펙션을 실행합니다.

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

--provide 옵션을 사용하여 인트로스펙션 완료 시 노드 상태를 **available**로 변경합니다.

6. **dnsmasq** 로그에서 노드의 **IP** 주소를 식별합니다.

```
(undercloud) $ sudo tail -f /var/log/containers/ironic-inspector/dnsmasq.log
```

7. 오류가 발생하면 **root** 사용자 및 임시 액세스 세부 정보를 사용하여 노드에 액세스합니다.

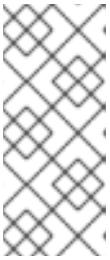
```
$ ssh root@192.168.24.105
```

인트로스펙션 중에 노드에 액세스하여 진단 명령을 실행하고 인트로스펙션 실패 문제를 해결합니다.

8. 인트로스펙션 프로세스를 중지하려면 다음 명령을 실행합니다.

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

프로세스가 시간 초과될 때까지 기다릴 수도 있습니다.



참고

Red Hat OpenStack Platform director는 초기 중단 후 인트로스펙션을 **3**번 재시도합니다. 각 시도에서 **openstack baremetal introspection abort** 명령을 실행하여 인트로스펙션을 완전히 중단합니다.

28.3. 워크플로우 및 실행 문제 해결

OpenStack Workflow(mistral) 서비스는 여러 **OpenStack** 작업을 워크플로우에 그룹화합니다. **Red Hat OpenStack Platform**은 이러한 워크플로우 세트를 사용하여 **director**를 통해 베어 메탈 노드 제어, 검증, 계획 관리, 오버클라우드 배포 등의 일반적인 기능을 수행합니다.

예를 들어 **openstack overcloud deploy** 명령을 실행할 때 **OpenStack Workflow** 서비스는 두 개의 워크플로우를 실행합니다. 첫 번째 워크플로우는 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 번째 워크플로우는 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
...
```

OpenStack Workflow 서비스는 다음 오브젝트를 사용하여 워크플로우를 추적합니다.

작업

관련 작업이 실행될 때 **OpenStack**에서 수행하는 특정한 명령입니다. 예를 들면 쉘 스크립트 실행 또는 **HTTP** 요청 수행이 있습니다. 일부 **OpenStack** 구성 요소에는 **OpenStack** 워크플로우에서 사용

하는 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.

OpenStack Workflow는 강력한 실행 기록 기능도 제공하므로 특정 명령이 실패했을 경우 문제를 식별하는 데 도움이 됩니다. 예를 들어 워크플로우 실행이 실패하는 경우 문제 지점을 확인할 수 있습니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 실패한 상태 **ERROR**가 있는 워크플로우 실행을 표시합니다.

```
(undercloud) $ openstack workflow execution list | grep "ERROR"
```

3. 실패한 워크플로우 실행의 **UUID(예: dffa96b0-f679-4cd2-a490-4769a3825262)**를 가져와서 실행 및 해당 출력을 살펴봅니다.

```
(undercloud) $ openstack workflow execution show dffa96b0-f679-4cd2-a490-4769a3825262
(undercloud) $ openstack workflow execution output show dffa96b0-f679-4cd2-a490-
4769a3825262
```

4. 이 명령을 실행하면 실행에 실패한 작업에 대한 정보가 반환됩니다. **openstack workflow execution show** 명령은 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.

5. 비슷한 명령 구문을 사용하여 작업 실행 및 해당 결과를 살펴봅니다.

```
(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
```

이 명령은 문제의 원인이 되는 특정 작업을 식별하는 데 유용합니다.

28.4. 오버클라우드 생성 및 배포 문제 해결

오버클라우드는 처음에 **OpenStack Orchestration(heat)** 서비스를 통해 생성됩니다. 오버클라우드 배포에 실패한 경우 **OpenStack** 클라이언트 및 서비스 로그 파일을 사용하여 실패한 배포를 진단합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 배포 실패 명령을 실행합니다.

```
$ openstack overcloud failures
```

3. 다음 명령을 실행하여 실패 세부 정보를 표시합니다.

```
(undercloud) $ openstack stack failures list <OVERCLOUD_NAME> --long
```

- **<OVERCLOUD_NAME>**을 해당 오버클라우드 이름으로 교체합니다.

4. 다음 명령을 실행하여 실패한 스택을 확인합니다.

```
(undercloud) $ openstack stack list --nested --property status=FAILED
```

28.5. 노드 프로비저닝 문제 해결

OpenStack Orchestration(heat) 서비스는 프로비저닝 프로세스를 제어합니다. 노드 프로비저닝에 실패한 경우 **OpenStack** 클라이언트 및 서비스 로그 파일을 사용하여 문제를 진단합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 베어 메탈 서비스를 검사하여 등록된 모든 노드와 노드의 현재 상태를 확인합니다.

```
(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       |
+----------+------+---------------+-------------+-----------------+-------------+
```

프로비저닝에 사용 가능한 모든 노드에서 다음 상태가 설정되어 있어야 합니다.

- **Maintenance**가 **False**로 설정
- 프로비저닝 전에 **Provision State**가 **available**로 설정

다음 표에는 몇 가지 일반적인 프로비저닝 실패 시나리오가 요약되어 있습니다.

문제	원인	해결 방법
Maintenance가 자동으로 **True**로 설정됩니다.	director가 노드의 전원 관리에 액세스할 수 없습니다.	노드 전원 관리에 사용되는 인증 정보를 확인합니다.
Provision State가 **available**로 설정되었지만 노드가 프로비저닝되지 않습니다.	베어 메탈 배포가 시작되기 전에 문제가 발생했습니다.	프로필 및 플레이버 매핑을 비롯한 노드 세부 정보를 확인합니다. 노드 하드웨어 세부 정보가 플레이버의 요구 사항에 맞는지 확인합니다.
노드의 Provision State가 **wait call-back**으로 설정되어 있습니다.	이 노드에 대한 노드 프로비저닝 프로세스가 아직 완료되지 않았습니다.	이 상태가 변경될 때까지 기다립니다. 또는 노드의 가상 콘솔에 연결하여 출력을 확인합니다.
Provision State가 **active**이고 Power State가 **power on**인데 노드가 응답하지 않습니다.	노드 프로비저닝이 성공적으로 완료되었으며 배포 후 설정 단계에서 문제가 발생했습니다.	노드 설정 프로세스를 진단합니다. 노드의 가상 콘솔에 연결하여 출력을 확인합니다.
Provision State가 **error** 또는 **deploy failed**입니다.	노드 프로비저닝에 실패했습니다.	**openstack baremetal node show** 명령을 사용하여 베어 메탈 노드 세부 정보를 살펴보고, 오류 설명이 포함된 **last_error** 필드를 확인합니다.

28.6. 프로비저닝 중 IP 주소 충돌 문제 해결

대상 호스트에 이미 사용 중인 IP 주소를 할당하면 인트로스펙션 및 배포 작업이 실패합니다. 이러한 오류를 방지하려면 프로비저닝 네트워크에 대해 포트 스캔을 수행하여 검색 IP 주소 범위와 호스트 IP 주소 범위가 사용 가능한지 확인합니다.

절차

1. **nmap**을 설치합니다.

```
$ sudo dnf install nmap
```

2. **nmap**을 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
```

3. **nmap** 스캔 출력을 검토합니다. 예를 들어 언더클라우드의 **IP** 주소 및 서브넷에 있는 다른 호스트가 모두 표시되어야 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
```

활성 **IP** 주소가 **undercloud.conf**의 **IP** 범위와 충돌하는 경우 **IP** 주소 범위를 변경하거나 **IP** 주소를 해제한 후에 오버클라우드 노드를 인트로스펙션하거나 배포해야 합니다.

28.7. "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.
```

이 오류는 **Compute** 스케줄러가 새 인스턴스 부팅에 적합한 베어 메탈 노드를 찾을 수 없는 경우에 발생합니다. 일반적으로 **Compute** 서비스가 찾아야 하는 리소스와 **Bare Metal** 서비스가 **Compute**에 알린 리소스가 일치하지 않음을 의미합니다. 불일치 오류가 있는지 확인하려면 다음 단계를 수행합니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 노드에서 인트로스펙션이 성공했는지 확인합니다. 인트로스펙션이 실패한 경우 각 노드에 필수 **ironic** 노드 속성이 포함되어 있는지 확인합니다.

```
(undercloud) $ openstack baremetal node show [NODE UUID]
```

properties JSON 필드에 **cpus, cpu_arch, memory_mb** 및 **local_gb** 키에 대한 올바른 값이 있는지 확인합니다.

3. 노드에 매핑된 **Compute** 플레이버가 필요한 노드 수의 노드 특성을 초과하지 않는지 확인합니다.

```
(undercloud) $ openstack flavor show [FLAVOR NAME]
```

4. **openstack baremetal node list** 명령을 실행하여 사용 가능 상태의 노드가 충분히 있는지 확인합니다. **manageable** 상태의 노드는 일반적으로 인트로스펙션 실패를 나타냅니다.

5. **openstack baremetal node list** 명령을 실행하고 노드가 유지보수 모드에 있지 않은지 확인합니다. 노드가 유지보수 모드로 자동 변경되는 경우 잘못된 전원 관리 인증 정보 문제 때문일 가능성이 큽니다. 전원 관리 인증 정보를 확인하고 유지보수 모드를 삭제합니다.

```
(undercloud) $ openstack baremetal node maintenance unset [NODE UUID]
```

6. 자동 프로필 태깅을 사용하는 경우 각 플레이버 및 프로필에 해당하는 노드가 충분한지 확인합니다. 노드에서 **openstack baremetal node show** 명령을 실행하고 **properties** 필드의 **capabilities** 키를 확인합니다. 예를 들어 **Compute** 역할에 태그된 노드에는 **profile:compute** 값이 들어 있습니다.

7. 인트로스펙션 후 노드 정보가 **Bare Metal**에서 **Compute**로 반영될 때까지 기다려야 합니다. 하지만 일부 단계를 수동으로 수행한 경우 **nova**에서 잠시 동안 노드를 **Compute** 서비스(**nova**)에 사용하지 못할 수 있습니다. 다음 명령을 사용하여 시스템의 총 리소스를 확인합니다.

```
(undercloud) $ openstack hypervisor stats show
```

28.8. 오버클라우드 설정 문제 해결

Red Hat OpenStack Platform director는 **Ansible**을 사용하여 오버클라우드를 설정합니다. 오버클라우드에서 **Ansible** 플레이북 오류(**config-download**)를 진단하려면 다음 단계를 수행합니다.

절차

1. **stack** 사용자가 **undercloud**의 **/var/lib/mistral** 디렉터리에 있는 파일에 액세스할 수 있는지 확인합니다.

```
$ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
```

이 명령을 수행하면 디렉터리에 대한 **mistral** 사용자의 액세스 권한도 유지됩니다.

2. **config-download** 파일의 작업 디렉터리로 변경합니다. 일반적으로 **/var/lib/mistral/overcloud/**입니다.

```
$ cd /var/lib/mistral/overcloud/
```

3. **ansible.log** 파일에서 오류 발생 지점을 검색합니다.

```
$ less ansible.log
```

오류 발생 단계를 적어 둡니다.

4. **config-download** 플레이북의 작업 디렉터리 내에서 오류가 발생한 단계를 찾아 발생한 작업을 확인합니다.

28.9. 컨테이너 설정 문제 해결

Red Hat OpenStack Platform director는 **paunch**를 사용하여 컨테이너를 실행하고 **podman**을 사용하여 컨테이너를 관리하며 **puppet**을 사용하여 컨테이너 설정을 생성합니다. 다음 절차에서는 오류가 발생했을 때 컨테이너를 진단하는 방법에 대해 설명합니다.

호스트 액세스

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 컨테이너 장애가 있는 노드의 **IP** 주소를 가져옵니다.

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

3. 노드에 로그인합니다.

```
(undercloud) $ ssh heat-admin@192.168.24.60
```

4. **root** 사용자로 변경합니다.

```
$ sudo -i
```

오류가 발생한 컨테이너 식별

1. 모든 컨테이너를 표시합니다.

```
$ podman ps --all
```

오류가 발생한 컨테이너를 식별합니다. 오류가 발생한 컨테이너는 일반적으로 **0**이외의 상태로 종료됩니다.

컨테이너 로그 확인

1. 각 컨테이너는 메인 프로세스의 표준 출력을 유지합니다. 이 출력을 로그로 사용하여 컨테이너 실행 중에 실제로 수행되는 작업을 확인합니다. 예를 들어 **keystone** 컨테이너의 로그를 보려면 다음 명령을 실행합니다.

```
$ sudo podman logs keystone
```

대부분의 경우 이 로그에는 컨테이너 오류 발생 원인에 대한 정보가 포함되어 있습니다.

2. 호스트는 실패한 서비스에 대한 **stdout** 로그도 유지합니다. **stdout** 로그는 **/var/log/containers/stdouts/**에서 찾을 수 있습니다. 예를 들어 오류가 발생한 **keystone** 컨테이너의 로그를 보려면 다음 명령을 실행합니다.

```
$ cat /var/log/containers/stdouts/keystone.log
```

컨테이너 검사

컨테이너 정보를 확인해야 하는 경우가 있습니다. 예를 들어 다음 명령을 사용하여 **keystone** 컨테이너 데이터를 확인합니다.

```
$ sudo podman inspect keystone
```

이 명령을 실행하면 하위 수준의 구성 데이터를 포함하는 **JSON** 오브젝트가 반환됩니다. 출력을 **jq** 명령으로 전달하여 특정 데이터를 구문 분석할 수 있습니다. 예를 들어 **keystone** 컨테이너에 대한 컨테이너 마운트를 보려면 다음 명령을 실행합니다.

```
$ sudo podman 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 podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}:{{ join .Options "," }}{{end}} -ti {{.Config.Image}}' keystone
```

참고

--format 옵션은 쿼리를 생성할 때 **Go** 구문을 사용합니다.

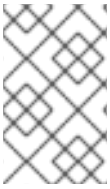
podman run 명령과 함께 이러한 옵션을 사용하면 문제 해결 목적으로 컨테이너를 재생성할 수 있습니다.

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo podman run --rm $OPTIONS /bin/bash
```

컨테이너에서 명령 실행

특정 **Bash** 명령을 통해 컨테이너 내부 정보를 가져와야 하는 경우가 있습니다. 이 경우에는 **podman** 명령을 사용하여 실행 중인 컨테이너 내에서 명령을 실행합니다. 예를 들어 **podman exec** 명령을 실행하여 **keystone** 컨테이너 내에서 명령을 실행합니다.

```
$ sudo podman exec -ti keystone <COMMAND>
```



참고

-ti 옵션은 대화식 가상 터미널**(pseudoterminal)**에서 명령을 실행합니다.

-

<COMMAND>를 실행하려는 명령으로 교체합니다. 예를 들어 각 컨테이너에는 서비스 연결을 확인하기 위한 상태 점검 스크립트가 있습니다. 다음 명령을 사용하여 keystone에 대해 상태 점검 스크립트를 실행할 수 있습니다.

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

컨테이너 쉘에 액세스하려면 컨테이너 내에서 실행하려는 명령으로 /bin/bash를 사용하여 podman exec를 실행합니다.

```
$ sudo podman exec -ti keystone /bin/bash
```

컨테이너 파일 시스템 확인

1. 오류가 발생한 컨테이너의 파일 시스템을 확인하려면 podman mount 명령을 실행합니다. 예를 들어 오류가 발생한 keystone 컨테이너의 파일 시스템을 확인하려면 다음 명령을 실행합니다.

```
$ podman mount keystone
```

이 명령을 실행하면 파일 시스템 내용을 볼 수 있는 마운트된 위치가 제공됩니다.

```
/var/lib/containers/storage/overlay/78946a109085aeb8b3a350fc20bd8049a08918d74f573396d7358270e711c610/merged
```

이 명령은 컨테이너 내에서 Puppet 보고서를 확인하는 데 유용합니다. 이 보고서는 컨테이너 마운트 내의 var/lib/puppet/ 디렉터리에서 찾을 수 있습니다.

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 podman export keystone -o keystone.tar
```

이 명령은 추출 및 확인할 수 있는 keystone.tar 압축 파일을 생성합니다.

28.10. 컴퓨팅 노드 장애 문제 해결

컴퓨팅 노드는 Compute 서비스를 사용하여 하이퍼바이저 기반 작업을 수행합니다. 따라서 컴퓨팅 노

드에 대한 주요 진단은 이 서비스와 관련이 있습니다.

절차

1. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

2. 장애가 있는 컴퓨팅 노드의 **IP** 주소를 가져옵니다.

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

3. 노드에 로그인합니다.

```
(undercloud) $ ssh heat-admin@192.168.24.60
```

4. **root** 사용자로 변경합니다.

```
$ sudo -i
```

5. 컨테이너 상태를 표시합니다.

```
$ sudo podman ps -f name=nova_compute
```

6. 컴퓨팅 노드의 주요 로그 파일은 **/var/log/containers/nova/nova-compute.log**입니다. 컴퓨팅 노드 통신에 문제가 발생하면 이 파일을 사용하여 진단을 시작합니다.

7. 컴퓨팅 노드에서 유지보수를 수행하는 경우 기존 인스턴스를 호스트에서 운영 가능한 컴퓨팅 노드로 마이그레이션한 다음, 해당 노드를 비활성화합니다.

28.11. SOSREPORT 생성

Red Hat OpenStack Platform에 대한 지원을 받기 위해 **Red Hat**에 문의하려는 경우 **sosreport**를 생성해야 할 수도 있습니다. **sosreport** 생성에 관한 자세한 내용은 다음을 참조하십시오.

-

"How to collect all required logs for Red Hat Support to investigate an OpenStack issue"

28.12. 로그 위치

문제를 해결할 때 다음 로그를 사용하여 언더클라우드 및 오버클라우드에 대한 정보를 수집합니다.

표 28.1. 언더클라우드 및 오버클라우드 노드의 로그

정보	로그 위치
컨테이너화된 서비스 로그	**/var/log/containers/**
컨테이너화된 서비스의 표준 출력	**/var/log/containers/stdouts**
Ansible 구성 로그	**/var/lib/mistral/overcloud/ansible.log**

표 28.2. 언더클라우드 노드의 추가 로그

정보	로그 위치
openstack overcloud deploy의 명령 내역	**/home/stack/.tripleo/history**
언더클라우드 설치 로그	**/home/stack/install-undercloud.log**

표 28.3. 오버클라우드 노드의 추가 로그

정보	로그 위치
Cloud-Init 로그	**/var/log/cloud-init.log**
고가용성 로그	**/var/log/pacemaker.log**

29장. 언더클라우드 및 오버클라우드 서비스 관련 팁

이 섹션에서는 언더클라우드의 특정 OpenStack 서비스 조정 및 관리와 관련된 정보를 제공합니다.

29.1. 배포 성능 조정

Red Hat OpenStack Platform director는 OpenStack Orchestration(heat)을 사용하여 주요 배포 및 프로비저닝 기능을 수행합니다. heat는 일련의 작업자를 사용하여 배포 작업을 실행합니다. 기본 작업자 수를 계산하기 위해 director의 heat 구성에서는 언더클라우드의 총 CPU 스레드 수를 반으로 나눕니다. 이 경우 스레드 수는 CPU 코어 수에 하이퍼 스레딩 값을 곱한 값입니다. 예를 들어 언더클라우드에 스레드가 16개인 CPU가 있는 경우, heat는 기본적으로 8개 작업자를 생성합니다. director 설정에서는 기본적으로 최소 및 최대 값을 사용합니다.

서비스	최소	최대
OpenStack Orchestration(heat)	4	24

그러나 환경 파일에서 HeatWorkers 매개변수를 사용하여 작업자 수를 수동으로 설정할 수 있습니다.

heat-workers.yaml

```
parameter_defaults:
  HeatWorkers: 16
```

undercloud.conf

```
custom_env_files: heat-workers.yaml
```

29.2. 컨테이너에서 SWIFT-RING-BUILDER 실행

Object Storage(swift) 링을 관리하려면 서버 컨테이너 내에서 swift-ring-builder 명령을 사용합니다.

- swift_object_server
- swift_container_server
- swift_account_server

예를 들어 swift 오브젝트 링에 대한 정보를 보려면 다음 명령을 실행합니다.

```
$ sudo podman exec -ti -u swift swift_object_server swift-ring-builder /etc/swift/object.builder
```

이 명령은 언더클라우드 및 오버클라우드 노드에서 모두 실행할 수 있습니다.

29.3. HAPROXY에 대한 SSL/TLS 암호화 규칙 변경

언더클라우드에서 SSL/TLS를 활성화한 경우 (4.2절. “director 설정 매개변수” 참조) HAProxy 구성에 사용되는 SSL/TLS 암호화 방식 및 규칙을 강화할 수 있습니다. 이러한 강화는 POODLE 취약점과 같은 SSL/TLS 취약점을 해결하는 데 도움이 됩니다.

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 다음과 같은 암호화 방식 및 규칙을 사용할 수 있습니다.

- 암호: 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:AES128-SHA256:A ES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS

- 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** 덮어쓰기 파일을 사용하도록 **undercloud.conf** 파일의 **hieradata_override** 매개변수를 설정합니다**.**

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```

30장. 전원 관리 드라이버

director는 기본적으로 전원 관리 컨트롤에 IPMI를 사용하지만 다른 전원 관리 유형도 지원합니다. 이 부록에는 director에서 지원하는 전원 관리 기능 목록이 포함되어 있습니다. 오버클라우드의 노드를 등록할 때 전원 관리 설정을 사용합니다. 자세한 내용은 오버클라우드 노드 등록을 참조하십시오.

30.1. IPMI(INTELLIGENT PLATFORM MANAGEMENT INTERFACE)

BMC(Baseboard Management Controller)를 사용할 경우 표준 전원 관리 방법입니다.

pm_type

이 옵션을 ipmi로 설정합니다.

pm_user; pm_password

IPMI 사용자 이름 및 암호입니다.

pm_addr

IPMI 컨트롤러의 IP 주소입니다.

pm_port(선택 사항)

IPMI 컨트롤러에 연결하는 포트입니다.

30.2. REDFISH

DMTF(Distributed Management Task Force)에서 개발한 IT 인프라를 위한 표준 RESTful API

pm_type

이 옵션을 redfish로 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 경로는 루트 서비스, 버전 및 시스템의 경로/유일 ID를 포함해야 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 false로 설정합니다. 그러나 BMC 인증을 비활성화하면 BMC에 대한 액세스 보안이 손상될 수 있습니다.

30.3. DRAC(DELL REMOTE ACCESS CONTROLLER)

DRAC는 전원 관리 및 서버 모니터링을 포함하여 대역 외 원격 관리 기능을 제공하는 인터페이스입니다.

pm_type

이 옵션을 idrac로 설정합니다.

pm_user; pm_password

DRAC 사용자 이름 및 암호입니다.

pm_addr

DRAC 호스트의 IP 주소입니다.

30.4. ILO(INTEGRATED LIGHTS-OUT)

Hewlett-Packard의 iLO는 전원 관리 및 서버 모니터링을 포함하여 대역 외 원격 관리 기능을 제공하는 인터페이스입니다.

pm_type

이 옵션을 ilo로 설정합니다.

pm_user; pm_password

iLO 사용자 이름 및 암호입니다.

pm_addr

iLO 인터페이스의 IP 주소입니다.

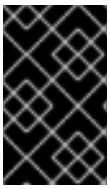
- 이 드라이버를 활성화하려면 **undercloud.conf**의 **enabled_hardware_types** 옵션에 **ilo**를 추가하고 **openstack undercloud install**을 재실행합니다.

- 인트로스펙션에 성공하려면 **HP** 노드에 최소 **ILO** 펌웨어 버전 **1.85(2015**년 **5**월 **13**일)가 있어야 합니다. **director**는이 **ILO** 펌웨어 버전을 사용하는 노드에서 성공적으로 테스트되었습니다.

- 공유 **iLO** 포트 사용은 지원되지 않습니다.

30.5. FUJITSU IRMC(INTEGRATED REMOTE MANAGEMENT CONTROLLER)

Fujitsu iRMC는 통합된 **LAN** 연결 및 확장된 기능이 있는 **BMC(Baseboard Management Controller)** 입니다. 이 드라이버는 **iRMC**에 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.

중요

iRMC 사용자에게 **ADMINISTRATOR** 권한이 있어야 합니다.

pm_addr

iRMC 인터페이스의 **IP** 주소입니다.

pm_port(선택 사항**)**

iRMC 작업의 포트입니다. 기본값은 **443**입니다.

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 **60**초입니다.

pm_sensor_method(선택 사항)

센서 데이터 검색 방법입니다. **ipmitool** 또는 **scci**를 사용합니다. 기본값은 **ipmitool**입니다.

- 이 드라이버를 활성화하려면 **undercloud.conf** 파일의 **enabled_hardware_types** 옵션에 **irmc** 를 추가하고 **openstack undercloud install** 명령을 재실행합니다.

UEFI 부팅 모드가 있는 **iRMC**

iRMC는 **UEFI** 모드로 부팅하려면 **ipxe.efi** 가 필요합니다. **UEFI**로 부팅하려면 **SNP(Simple Network Protocol) iPXE EFI**를 사용하는 기본 동작을 비활성화하고 언더클라우드를 다시 설치합니다.

절차

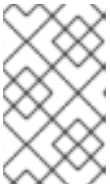
1. 사용자 지정 환경 파일(예: **/home/stack/templates/irmc-uefi-boot.yaml**)을 생성합니다.

2. 사용자 지정 환경 파일에 다음 구성을 추가합니다.

```
parameter_defaults:
  IronicIPXEUefiSnpOnly: false
```

3. **undercloud.conf** 파일에서 **custom_env_files** 매개변수를 편집하여 사용자 지정 환경 파일을 추가합니다.

```
custom_env_files = /home/stack/templates/irmc-uefi-boot.yaml
```



참고

쉼표로 구분된 목록을 사용하여 여러 환경 파일을 지정할 수 있습니다.

4. 언더클라우드를 다시 설치하여 설정 업데이트를 적용합니다.

```
$ OpenStack undercloud install
```

30.6. RED HAT VIRTUALIZATION

이 드라이버는 **RESTful API**를 통해 **RHV(Red Hat Virtualization)**에서 가상 머신을 제어합니다.

pm_type

이 옵션을 **staging-ovirt**로 설정합니다.

pm_user; pm_password

RHV 환경의 사용자 이름과 암호입니다. 사용자 이름에는 인증 제공업체 또한 포함되어 있습니다. 예: **admin@internal**

pm_addr

RHV REST API의 **IP** 주소입니다.

pm_vm_name

제어할 가상 머신의 이름입니다.

mac

노드에 있는 네트워크 인터페이스의 **MAC** 주소 목록입니다. 각 시스템의 프로비저닝 **NIC**에는 **MAC** 주소만 사용합니다.

- 이 드라이버를 활성화하려면 **undercloud.conf**의 **enabled_hardware_types** 옵션에 **staging-ovirt**를 추가하고 **openstack undercloud install** 명령을 재실행합니다.

30.7. MANUAL-MANAGEMENT 드라이버

전원 관리 기능이 없는 베어 메탈 장치를 제어하려면 **manual-management** 드라이버를 사용합니다. **director**는 등록된 베어 메탈 장치를 제어하지 않으므로, 인트로스펙션 및 배포 프로세스의 특정 시점에서 수동 전원 관리 작업을 수행해야 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 **manual-management**로 설정합니다.

- 이 드라이버는 전원 관리를 제어하지 않으므로 인증 세부 정보를 사용하지 않습니다.
- 이 드라이버를 활성화하려면 **undercloud.conf**의 **enabled_hardware_types** 옵션에 **manual-management**를 추가하고 **openstack undercloud install** 명령을 재실행합니다.
- **instackenv.json** 노드 인벤토리 파일에서 수동으로 관리하려는 노드에 대해 **pm_type**을 **manual-management**로 설정합니다.

인트로스펙션

- 노드에서 인트로스펙션을 수행하는 경우, **openstack overcloud node introspect** 명령을 실행한 후 노드를 수동으로 시작합니다. 노드가 **PXE**를 통해 부팅되는지 확인합니다.
- 노드 정리를 활성화한 경우 인트로스펙션 완료 메시지가 표시된 후 노드를 수동으로 재부팅하고, **openstack baremetal node list** 명령을 실행할 때 노드 상태가 각 노드를 완전히 대기 합니다. 노드가 **PXE**를 통해 부팅되는지 확인합니다.
- 인트로스펙션 및 정리 프로세스가 완료되면 노드를 종료합니다.

Deployment

- 오버클라우드 배포를 수행할 때 **openstack baremetal node list** 명령을 사용하여 노드 상태를 확인합니다. 노드 상태가 **deploying** 에서 **call-back** 으로 변경될 때까지 기다린 다음 노드를

수동으로 시작합니다. 노드가 **PXE**를 통해 부팅되는지 확인합니다.

- 오버클라우드 프로비저닝 프로세스가 완료되면 노드가 종료됩니다. 구성 프로세스를 시작하려면 디스크에서 노드를 부팅해야 합니다. 프로비저닝이 완료되었는지 확인하려면 **openstack baremetal node list** 명령을 사용하여 노드 상태를 확인하고 노드 상태가 각 노드에 대해 활성 상태로 변경될 때까지 기다립니다. 노드 상태가 **active** 인 경우 프로비저닝된 오버클라우드 노드를 수동으로 부팅합니다.