



Red Hat OpenStack Platform 11

Director 설치 및 사용

Red Hat OpenStack Platform director를 사용하여 OpenStack 클라우드 환경을 생성을
위한 종단간 시나리오

Red Hat OpenStack Platform 11 Director 설치 및 사용

Red Hat OpenStack Platform director를 사용하여 OpenStack 클라우드 환경 생성을 위한 종단간 시나리오

OpenStack Team
rhos-docs@redhat.com

법적 공지

Copyright © 2018 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

초록

본 가이드에서는 Red Hat OpenStack Platform Director를 사용하는 엔터프라이즈 환경에 Red Hat OpenStack Platform 11을 설치하는 방법을 설명합니다. 여기에는 **director** 설치, 환경 계획, **director**를 사용한 OpenStack 환경 구축 작업이 포함됩니다.

차례

1장. 소개	6
1.1. 언더클라우드	6
1.2. 오버클라우드	7
1.3. 고가용성	9
1.4. CEPH STORAGE	9
2장. 요구 사항	10
2.1. 환경 요구 사항	10
2.2. 언더클라우드 요구 사항	10
2.2.1. 가상화 지원	11
2.3. 네트워킹 요구 사항	13
2.4. 오버클라우드 요구 사항	15
2.4.1. Compute 노드 요구 사항	15
2.4.2. Controller 노드 요구 사항	16
2.4.3. Ceph Storage 노드 요구 사항	17
2.4.4. Object Storage 노드 요구 사항	17
2.5. 리포지토리 요구 사항	18
3장. 오버클라우드 플래닝	20
3.1. 노드 배포 역할 플래닝	20
3.2. 네트워크 플래닝	21
3.3. 스토리지 플래닝	25
4장. 언더클라우드 설치	27
4.1. DIRECTOR 설치 사용자 생성	27
4.2. 템플릿 및 이미지 용 디렉터리 생성	27
4.3. 시스템의 호스트 이름 설정	27
4.4. 시스템 등록	28
4.5. DIRECTOR 패키지 설치	29
4.6. DIRECTOR 구성	29
4.7. 오버클라우드 노드의 이미지 가져오기	33
4.8. 언더클라우드의 NEUTRON 서브넷에서 네임 서버 설정	35
4.9. 언더클라우드 백업	35
4.10. 언더클라우드 설정 완료	35
5장. CLI 툴로 기본 오버클라우드 설정	36
5.1. 오버클라우드에 노드 등록	36
5.2. 노드의 하드웨어 검사	38
5.3. 프로필에 노드 태그	39
5.4. 노드의 ROOT 디스크 정의	40
5.5. 오버클라우드 사용자 정의	42
5.6. CLI 툴로 오버클라우드 생성	42
5.7. 오버클라우드 생성에 환경 파일 추가	47
5.8. 오버클라우드 플랜 관리	49
5.9. 오버클라우드 템플릿 및 플랜 검증	50
5.10. 오버클라우드 생성 모니터링	50
5.11. 오버클라우드 액세스	51
5.12. 오버클라우드 생성 완료	51
6장. 웹 UI로 기본 오버클라우드 설정	52
6.1. 웹 UI에 액세스	52
6.2. 웹 UI 탐색	54
6.3. 웹 UI에서 오버클라우드 플랜 가져오기	55

6.4. 웹 UI에서 노드 등록	56
6.5. 웹 UI에서 노드의 하드웨어 검사	58
6.6. 웹 UI에서 오버클라우드 플랜 매개 변수 편집	59
6.7. 웹 UI에서 역할에 노드 태그	60
6.8. 웹 UI에서 노드 편집	61
6.9. 웹 UI에서 오버클라우드 생성 시작	64
6.10. 오버클라우드 생성 완료	65
7장. 사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 구성	66
7.1. 노드 구성을 위한 사용자 생성	67
7.2. 노드의 운영 체제 등록	67
7.3. 노드에 사용자 에이전트 설치	68
7.4. DIRECTOR에 대한 SSL/TLS 액세스 구성	68
7.5. 컨트롤 플레인에 대한 네트워킹 구성	69
7.6. 오버클라우드 노드에 별도의 네트워크 사용	70
7.7. 사전 프로비저닝된 노드로 오버클라우드 생성	72
7.8. 메타데이터 서버 폴링	74
7.9. 오버클라우드 생성 모니터링	76
7.10. 오버클라우드 액세스	76
7.11. 사전 프로비저닝된 노드 확장	76
7.12. 사전 프로비저닝된 오버클라우드 제거	77
7.13. 오버클라우드 생성 완료	77
8장. 오버클라우드 생성 후 작업 수행	78
8.1. 오버클라우드 테넌트 네트워크 생성	78
8.2. 오버클라우드 외부 네트워크 생성	78
8.3. 추가 유동 IP 네트워크 생성	79
8.4. 오버클라우드 공급자 네트워크 생성	80
8.5. 오버클라우드 검증	80
8.6. 오버클라우드 환경 수정	81
8.7. 가상 머신을 오버클라우드로 가져오기	82
8.8. 오버클라우드 COMPUTE 노드에서 VM 마이그레이션	82
8.9. ANSIBLE 자동화 실행	83
8.10. 오버클라우드 삭제 방지	85
8.11. 오버클라우드 제거	85
9장. 오버클라우드 확장	86
9.1. 기타 노드 추가	86
9.2. COMPUTE 노드 제거	88
9.3. COMPUTE 노드 교체	89
9.4. CONTROLLER 노드 교체	90
9.4.1. 사전 점검	90
9.4.2. 노드 교체	91
9.4.3. 수동 조작	93
9.4.4. 오버클라우드 서비스 구성 완료	98
9.4.5. L3 에이전트 라우터 호스팅 구성 완료	99
9.4.6. Compute 서비스 구성 완료	99
9.4.7. 결론	100
9.5. CEPH STORAGE 노드 교체	100
9.6. OBJECT STORAGE 노드 교체	100
10장. 노드 재부팅	102
10.1. DIRECTOR 재부팅	102
10.2. CONTROLLER 노드 재부팅	102

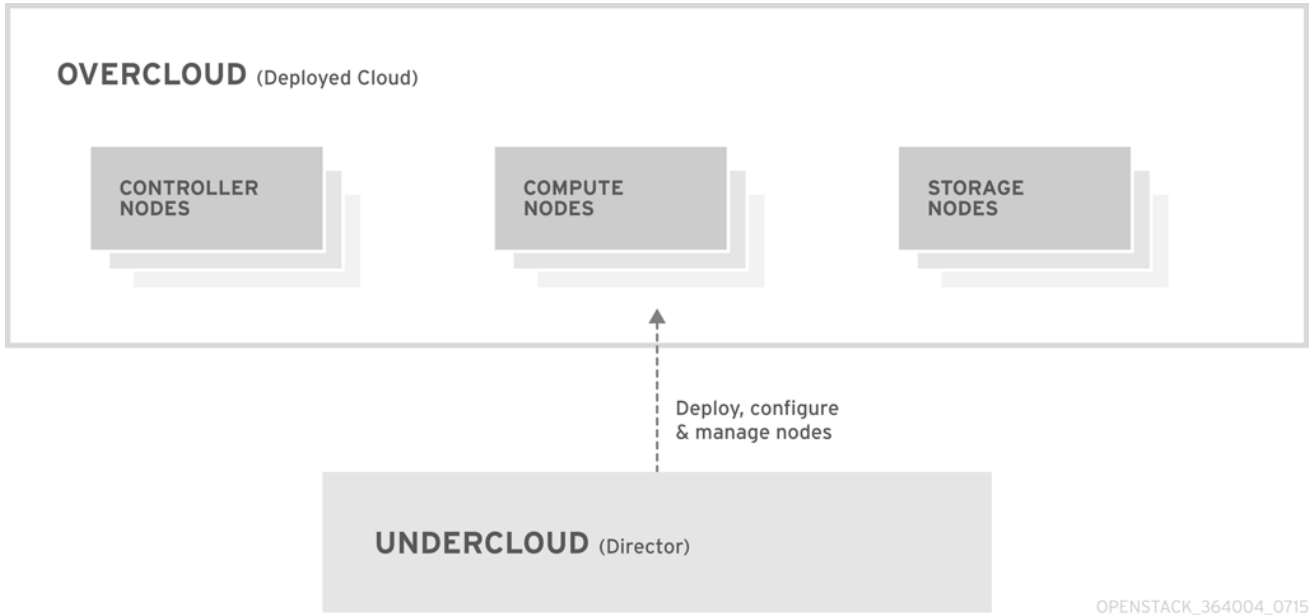
10.3. CEPH STORAGE 노드 재부팅	103
10.4. COMPUTE 노드 재부팅	104
10.5. OBJECT STORAGE 노드 재부팅	105
11장. DIRECTOR 문제 해결	106
11.1. 노드 등록 문제 해결	106
11.2. 하드웨어 INTROSPECTION 문제 해결	106
11.3. 워크플로우 및 실행 문제 해결	108
11.4. 오버클라우드 생성 문제 해결	109
11.4.1. 오케스트레이션	109
11.4.2. 베어 메탈 프로비저닝	110
11.4.3. 배포 후 구성	111
11.5. 프로비저닝 네트워크에서 IP 주소 충돌 문제 해결	112
11.6. "NO VALID HOST FOUND" 오류 문제 해결	113
11.7. 오버클라우드 생성 후 문제 해결	114
11.7.1. 오버클라우드 스택 수정	114
11.7.2. Controller 서비스 오류	115
11.7.3. Compute 서비스 오류	115
11.7.4. Ceph Storage 서비스 오류	116
11.8. 언더클라우드 성능 조정	116
11.9. 언더클라우드 및 오버클라우드 관련 중요 로그	117
부록 A. SSL/TLS 인증서 구성	119
A.1. 서명 호스트 초기화	119
A.2. 인증 기관 생성	119
A.3. 클라이언트에 인증 기관 추가	119
A.4. SSL/TLS 키 생성	119
A.5. SSL/TLS 인증서 서명 요청 생성	120
A.6. SSL/TLS 인증서 생성	121
A.7. 언더클라우드에서 인증서 사용	121
부록 B. 전원 관리 드라이버	123
B.1. DRAC(DELL REMOTE ACCESS CONTROLLER)	123
B.2. ILO(INTEGRATED LIGHTS-OUT)	123
B.3. IBOOT	123
B.4. CISCO UCS(UNIFIED COMPUTING SYSTEM)	124
B.5. FUJITSU IRMC(INTEGRATED REMOTE MANAGEMENT CONTROLLER)	124
B.6. VBM(VIRTUAL BARE METAL CONTROLLER)	125
B.7. SSH 및 VIRSH	127
B.8. 페이크 PXE 드라이버	128
부록 C. 전체 디스크 이미지	129
C.1. 전체 디스크 이미지 생성	129
C.2. 전체 디스크 이미지를 수동으로 생성	129
C.3. 자동으로 전체 디스크 이미지 생성	130
C.4. 전체 디스크 이미지의 블록 암호화	133
C.5. 전체 디스크 이미지 업로드	136
부록 D. 대체 부팅 모드	137
D.1. 표준 PXE	137
D.2. UEFI 부팅 모드	137
부록 E. 자동 프로파일 태깅	138
E.1. 정책 파일 구문	138
E.2. 정책 파일 예	139

E.3. 정책 파일 가져오기	141
E.4. 자동 프로필 태깅 속성	142
부록 F. 보안 강화	143
F.1. HAPROXY에 대한 SSL/TLS 암호화 방식 및 규칙 변경	143

1장. 소개

Red Hat OpenStack Platform director는 전체 OpenStack 환경을 설치하고 관리하는 툴셋으로, 주로 OpenStack 프로젝트 TripleO ("OpenStack-On-OpenStack"의 약어)를 기반으로 합니다. 이 프로젝트는 OpenStack 구성 요소를 이용하여 완전하게 기능하는 OpenStack 환경을 설치합니다. 여기에는 OpenStack 노드로 사용할 베어 메탈 시스템을 프로비저닝하고 제어하는 새로운 OpenStack 구성 요소가 포함되며, 이를 통해 간단하면서도 강력한 전체 Red Hat OpenStack Platform 환경을 간편하게 설치할 수 있습니다.

Red Hat OpenStack Platform director는 언더클라우드(undercloud)와 오버클라우드(overcloud)의 두 가지 주요 개념을 사용합니다. 언더클라우드에서 오버클라우드를 설치 및 구성합니다. 다음 부분에서는 각각의 개념에 대해 간략하게 설명합니다.



OPENSTACK_364004_0715

1.1. 언더클라우드

언더클라우드는 주요 director 노드로 OpenStack 환경(오버클라우드)을 구성하는 OpenStack 노드를 프로비저닝하고 관리하는 구성 요소가 포함된 단일 시스템 OpenStack 설치 노드입니다. 언더클라우드의 구성 요소는 다음과 같은 여러 기능을 제공합니다.

환경 플래닝

언더클라우드는 사용자가 특정 노드 역할을 생성하고 할당할 수 있는 플래닝 기능을 제공합니다. 언더클라우드에는 Compute, Controller 및 여러 스토리지 역할과 같은 기본 노드 세트가 포함되어 있을 뿐만 아니라 사용자 정의 역할을 사용하는 기능도 제공합니다. 또한 각 노드 역할에 포함할 OpenStack Platform 서비스를 선택하여 새로운 노드 유형을 모델링하거나 해당 호스트의 특정 구성 요소를 분리하는 방법을 제공합니다.

베어 메탈 시스템 컨트롤

언더클라우드는 전원 관리 컨트롤 및 PXE 기반 서비스에서 하드웨어 속성을 검색하고 OpenStack을 각 노드에 설치하는 데 각 노드의 대역 외 관리 인터페이스(일반적으로 IPMI: Intelligent Platform Management Interface)를 사용합니다. 따라서 베어 메탈 시스템을 OpenStack 노드로 프로비저닝할 수 있습니다. 전원 관리 드라이버에 대한 전체 목록은 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

오케스트레이션

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

명령줄 툴 및 웹 UI

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

언더클라우드 구성 요소

언더클라우드는 OpenStack 구성 요소를 해당 기본 툴셋으로 사용합니다. 여기에는 다음과 같은 구성 요소가 포함되어 있습니다.

- OpenStack Identity(keystone) - director의 구성 요소에 인증 및 권한 부여를 제공합니다.
- OpenStack Bare Metal(ironic) 및 OpenStack Compute(nova) - 베어 메탈 노드를 관리합니다.
- OpenStack Networking(neutron) 및 Open vSwitch - 베어 메탈 노드에 대한 네트워킹을 제어합니다.
- OpenStack Image Service(glance) - 베어 메탈 머신에 기록된 이미지를 저장합니다.
- OpenStack Orchestration(heat) 및 Puppet - director에서 오버클라우드 이미지를 디스크에 기록한 후 노드의 오케스트레이션 및 노드 설정을 제공합니다.
- OpenStack Telemetry(ceilometer) - 모니터링 및 데이터 수집을 수행합니다. 여기에는 다음이 포함됩니다.
 - OpenStack Telemetry Metrics(Gnocchi) - 메트릭에 시계열 데이터베이스를 제공
 - OpenStack Telemetry Alarming(aodh) - 모니터링을 위한 알람 구성 요소를 제공
 - OpenStack Telemetry Event Storage(panko) - 모니터링을 위한 이벤트 스토리지를 제공
- OpenStack Workflow Service(mistral) - 플랜 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 Service용 이미지 스토리지
 - OpenStack Bare Metal에 대한 내부 검사 데이터
 - OpenStack Workflow Service에 대한 배포 플랜

1.2. 오버클라우드

오버클라우드는 언더클라우드를 사용하여 생성된 Red Hat OpenStack Platform 환경으로 OpenStack Platform 환경을 기반으로 다양한 노드 역할이 포함됩니다. 언더클라우드에는 다음과 같은 기본 오버클라우드 노드 역할이 포함됩니다.

Controller

OpenStack 환경에 관리, 네트워킹 및 고가용성 기능을 제공하는 노드입니다. 이상적인 OpenStack 환경은 고가용성 클러스터에 이러한 세 가지 노드를 함께 사용하는 것입니다.

기본 Controller 노드에는 다음 구성 요소가 포함됩니다.

- OpenStack Dashboard(horizon)

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

Compute

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

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

Storage

OpenStack 환경에 스토리지를 제공하는 노드로 다음과 같은 노드가 포함됩니다.

- **Ceph Storage** 노드 - 스토리지 클러스터를 만드는 데 사용됩니다. 각 노드에는 **Ceph OSD(Object Storage Daemon)**가 포함됩니다. 또한, **director**는 **Ceph Storage** 노드를 배포하는 **Controller** 노드에 **Ceph Monitor**를 설치합니다.
- **Block Storage(cinder)** - **HA Controller** 노드에 대한 외부 블록 스토리지로 사용됩니다. 이 노드에는 다음 구성 요소가 포함됩니다.
 - OpenStack Block Storage(cinder) 볼륨
 - OpenStack Telemetry(ceilometer) 에이전트

- Open vSwitch
- **Object Storage(swift)** - 이러한 노드는 **Openstack Swift**에 외부 스토리지 계층을 제공합니다. **Controller** 노드는 **Swift** 프록시를 통해 이러한 노드에 액세스합니다. 이 노드에는 다음 구성 요소가 포함되어 있습니다.
 - OpenStack Object Storage(swift) 스토리지
 - OpenStack Telemetry(ceilometer) 에이전트
 - Open vSwitch

1.3. 고가용성

Red Hat OpenStack Platform director는 **Controller** 노드 클러스터를 사용하여 **OpenStack Platform** 환경에 고가용성 서비스를 제공합니다. **director**에서는 각 **Controller** 노드에 중복 구성 요소 세트를 설치하여 하나의 단일 서비스로 관리합니다. 이 유형의 클러스터 구성은 단일 **Controller** 노드에 작동 오류가 발생하는 경우 폴백 기능을 제공하므로 **OpenStack** 사용자에게 어느 정도의 중단없는 서비스가 제공됩니다.

OpenStack Platform director는 일부 주요 소프트웨어를 사용하여 **Controller** 노드에서 구성 요소를 관리합니다.

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



참고

Red Hat OpenStack Platform director는 **Controller** 노드에서 대부분의 고가용성 설정을 자동으로 구성합니다. 하지만 전원 관리 컨트롤을 사용하도록 설정하려면 노드에 몇 가지 수동 구성이 필요합니다. 본 가이드에서는 이러한 절차를 설명하고 있습니다.

1.4. CEPH STORAGE

OpenStack을 사용하는 대규모 조직에서는 일반적으로 수많은 클라이언트를 지원합니다. 각 **OpenStack** 클라이언트에는 블록 스토리지 리소스를 사용할 때 고유한 요구 사항이 있을 수 있습니다. **glance**(이미지), **cinder**(블록) 및/또는 **nova**(Compute)를 단일 노드에 배포하면 수많은 클라이언트가 있는 대규모 배포인 경우 관리가 불가능해질 수 있습니다. 이러한 문제는 **OpenStack**을 외부적으로 확장하여 해결할 수 있습니다.

하지만 실질적으로는 **Red Hat OpenStack Platform** 스토리지 계층을 수십 테라바이트에서 펨타바이트(또는 엑사바이트) 스토리지로 확장할 수 있도록 **Red Hat Ceph Storage**와 같은 솔루션을 사용하여 스토리지 계층을 가상화해야 하는 요구 사항도 존재합니다. **Red Hat Ceph Storage**는 상용 하드웨어에서 실행되는 동안 이 스토리지 가상화 계층에 고가용성 및 고성능 기능을 제공합니다. 가상화에는 성능 저하가 나타나는 것처럼 보일 수 있지만, **Ceph** 스트라이프는 클러스터에서 오브젝트로서의 장치 이미지를 차단합니다. 이는 큰 **Ceph** 블록 장치 이미지가 독립 실행형 디스크보다 성능이 뛰어나다는 것을 의미합니다. 또한 **Ceph** 블록 장치는 성능 향상을 위해 캐싱, **copy-on-write** 복제 및 **copy-on-read** 복제를 지원합니다.

Red Hat Ceph Storage에 대한 자세한 내용은 [Red Hat Ceph Storage](#) 를 참조하십시오.

2장. 요구 사항

이 장에서는 **director**를 사용하여 **Red Hat OpenStack Platform**을 프로비저닝할 환경을 설정하기 위한 기본 요구 사항을 간략히 설명합니다. 여기에는 **director** 설정 및 액세스를 위한 요구 사항과 **director**에서 **OpenStack** 서비스를 위해 프로비저닝하는 호스트에 대한 하드웨어 요구 사항이 포함됩니다.



참고

Red Hat OpenStack Platform을 배포하기 전에 가능한 배포 방법의 특성을 고려하는 것이 중요합니다. 자세한 내용은 [Red Hat OpenStack Platform 설치 및 관리](#)를 참조하십시오.

2.1. 환경 요구 사항

최소 요구 사항:

- Red Hat OpenStack Platform director용 호스트 머신 한 대
- Red Hat OpenStack Platform Compute 노드용 호스트 머신 한 대
- Red Hat OpenStack Platform Controller 노드용 호스트 머신 한 대

권장 요구 사항:

- Red Hat OpenStack Platform director용 호스트 머신 한 대
- Red Hat OpenStack Platform Compute 노드용 호스트 머신 세 대
- 클러스터의 Red Hat OpenStack Platform Controller 노드용 호스트 머신 세 대
- 클러스터의 Red Hat Ceph Storage 노드용 호스트 머신 세 대

다음 사항에 유의하십시오.

- 모든 노드에 베어 메탈 시스템을 사용하는 것이 좋습니다. 적어도 **Compute** 노드에는 베어 메탈 시스템을 사용해야 합니다.
- 모든 오버클라우드 베어 메탈 시스템에는 **IPMI(Intelligent Platform Management Interface)**가 있어야 합니다. 이는 **director**가 전원 관리를 제어하기 때문입니다.



주의

Open vSwitch(OVS) 2.4.0에서 **OVS 2.5.0**으로 업그레이드하지 않은 경우 **Red Hat Enterprise Linux 7.3** 커널로 업그레이드하지 마십시오. 커널만 업그레이드하는 경우 **OVS**의 작동이 중지됩니다.

2.2. 언더클라우드 요구 사항

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

- Intel 64 또는 AMD64 CPU 확장 기능을 지원하는 8코어 64비트 x86 프로세서
- 최소 16GB RAM
- root 디스크에서 최소 40GB의 사용 가능한 디스크 공간. 오버클라우드 배포 또는 업데이트를 시도하려면 최소 10GB의 사용 가능한 공간이 있어야 합니다. 이러한 여유 공간은 노드 프로비저닝 프로세스 동안 이미지 변환 및 캐싱에 사용됩니다.
- 최소 2개의 1Gbps 네트워크 인터페이스 카드. 특히, 오버클라우드 환경에서 많은 수의 노드를 프로비저닝하는 경우 프로비저닝 네트워크 트래픽에 10Gbps 인터페이스를 사용하는 것이 좋습니다.
- 호스트 운영 체제로 설치된 Red Hat Enterprise Linux 7.3
- 호스트에서 활성화된 SELinux

2.2.1. 가상화 지원

Red Hat은 다음 플랫폼에서 가상화된 언더클라우드만 지원합니다.

플랫폼	비고
KVM(Kernel-based Virtual Machine)	인증된 하이퍼바이저에 나열된 대로 Red Hat Enterprise Linux 5, 6, 7에서 호스팅
Red Hat Enterprise Virtualization	인증된 하이퍼바이저에 나열된 대로 Red Hat Enterprise Virtualization 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 4.0에서 호스팅
Microsoft Hyper-V	Red Hat Customer Portal Certification Catalogue 에 나열된 대로 Hyper-V 버전에서 호스팅
VMware ESX 및 ESXi	Red Hat Customer Portal Certification Catalogue 에 나열된 대로 ESX 및 ESXi 버전에서 호스팅



중요

Red Hat OpenStack Platform director 11을 사용하려면 호스트 운영 체제로 최신 버전의 Red Hat Enterprise Linux를 사용해야 합니다. 즉, 가상화 플랫폼에서 기존 Red Hat Enterprise Linux 버전도 지원해야 합니다.

가상 머신 요구 사항

가상 언더클라우드에 대한 리소스 요구 사항은 베어 메탈 언더클라우드의 리소스 요구 사항과 비슷합니다. 프로비저닝할 때 네트워크 모델, 게스트 CPU 기능, 스토리지 백엔드, 스토리지 포맷 및 캐싱 모드와 같은 여러 튜닝 옵션을 고려해야 합니다.

네트워크 고려 사항

가상화된 언더클라우드에 대한 다음 네트워크 고려 사항을 확인하십시오.

전원 관리

언더클라우드 VM에서는 오버클라우드 노드의 전원 관리 장치에 액세스할 수 있어야 합니다. 이는 노드를 등록할 때 **pm_addr** 매개 변수에 대해 설정된 IP 주소입니다.

프로비저닝 네트워크

프로비저닝 네트워크(**ctlplane**)에 사용된 NIC에는 오버클라우드의 베어 메탈 노드의 NIC로 DHCP 요청을 브로드캐스트하고 처리하는 기능이 필요합니다. VM의 NIC를 베어 메탈 NIC와 동일한 네트워크에 연결하는 브리지를 생성하는 것이 좋습니다.



참고

하이퍼바이저 기술로 인해 언더클라우드가 알 수 없는 주소의 트래픽을 전송하지 못하는 경우 일반적으로 문제가 발생합니다. Red Hat Enterprise Virtualization을 사용하는 경우 **anti-mac-spoofing**을 비활성화하여 이 문제가 발생하지 않도록 하십시오. VMware ESX 또는 ESXi를 사용하는 경우 위장된 전송을 허용하여 이를 방지하십시오.

아키텍처 예

다음은 KVM 서버를 사용하는 기본 언더클라우드 가상화 아키텍처의 예로, 네트워크 및 리소스 요구 사항에 따라 빌드할 수 있는 기반으로서의 사용을 목적으로 하고 있습니다.

KVM 호스트는 다음 두 가지 Linux 브리지를 사용합니다.

br-ex(eth0)

- 외부에서 언더클라우드에 액세스
- 외부 네트워크의 DHCP 서버에서 가상 NIC(eth0)를 사용하여 언더클라우드에 네트워크 설정을 지정
- 언더클라우드에서 베어 메탈 서버의 전원 관리 인터페이스에 액세스할 수 있는 권한 제공

br-ctlplane(eth1)

- 베어 메탈 오버클라우드 노드와 동일한 네트워크에 연결
- 언더클라우드가 가상 NIC(eth1)를 통해 DHCP 및 PXE 부팅 요청 이행
- 오버클라우드의 베어 메탈 서버는 네트워크에서 PXE를 통해 부팅을 수행

KVM 호스트에는 다음 패키지가 필요합니다.

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

다음 명령은 KVM 호스트에 언더클라우드 가상 머신을 생성하고 해당 브리지에 연결하는 두 개의 가상 NIC를 생성합니다.

```
$ virt-install --name undercloud --memory=16384 --vcpus=4 --location /var/lib/libvirt/images/rhel-server-7.3-x86_64-dvd.iso --disk size=100 --network bridge=br-ex --network bridge=br-ctlplane --graphics=vnc --hvm --os-variant=rhel7
```

이 명령을 통해 **libvirt** 도메인이 시작됩니다. **virt-manager**를 사용하여 이 도메인에 연결하고 설치 프로세스를 진행합니다. 또는 다음 옵션을 사용하여 **kickstart** 파일을 통해 무인 설치를 수행할 수 있습니다.


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

설치가 완료되면 **root** 사용자로 인스턴스에 SSH 연결하여 [4장. 언더클라우드 설치](#)의 지침을 따르십시오.

백업

가상화된 언더클라우드를 백업하기 위한 여러 솔루션이 있습니다.

- **옵션 1:** [Director 언더클라우드 백업 및 복원](#) 가이드의 지침을 따릅니다.
- **옵션 2:** 언더클라우드를 종료하고 언더클라우드 가상 머신 스토리지의 백업 사본을 만듭니다.
- **옵션 3:** 하이퍼바이저가 라이브 또는 아토믹 스냅샷을 지원하는 경우 언더클라우드 VM의 스냅샷을 만듭니다.

KVM 서버를 사용하는 경우 다음 절차를 사용하여 스냅샷을 만듭니다.

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

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

1. QCOW 백업 파일의 복사본 (읽기 전용)을 만듭니다.

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

1. QCOW 오버레이 파일을 백업 파일에 병합하고, 원본 파일을 사용하도록 언더클라우드 VM을 다시 전환합니다.

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

2.3. 네트워킹 요구 사항

언더클라우드 호스트에는 최소 두 개의 네트워크가 필요합니다.

- **프로비저닝 네트워크** - 오버클라우드에서 사용할 베어 메탈 시스템을 찾을 수 있도록 DHCP 및 PXE 부팅 기능을 제공합니다. 일반적으로 이 네트워크는 **director**가 PXE 부팅 및 DHCP 요구에 대응할 수 있도록 트렁크된 인터페이스에서 기본 VLAN을 사용해야 합니다. 일부 서버 하드웨어 BIOS는 VLAN에서의 PXE 부팅을 지원하지만, BIOS가 부팅 후 기본 VLAN으로 해당 VLAN의 변환 기능을 지원해야 합니다. 이 기능이 지원되지 않는 경우 언더클라우드에 연결할 수 없습니다. 현재는 서버 하드웨어의 일부 하위 집합만 이 기능을 지원합니다. 이 네트워크는 모든 오버클라우드 노드에서 IPMI(Intelligent Platform Management Interface)를 통해 전원 관리를 제어하는 데 사용하는 네트워크이기도 합니다.
- **외부 네트워크** - 모든 노드에 원격 연결에 사용되는 별도의 네트워크입니다. 이 네트워크에 연결하는 인터페이스에는 외부 DHCP 서비스를 통해 정적 또는 동적으로 정의된 라우팅 가능한 IP 주소가 필요합니다.

이는 필요한 최소 네트워크 수를 나타냅니다. 하지만, **director**가 다른 Red Hat OpenStack Platform 네트워크 트래픽을 다른 네트워크로 분리할 수 있습니다. Red Hat OpenStack Platform은 네트워크 분리에 실제 인터페이스와 태그된 VLAN을 모두 지원합니다.

다음 사항에 유의하십시오.

- 일반적인 최소 오버클라우드 네트워크 구성은 다음과 같습니다.
 - 단일 NIC 구성 - 기본 VLAN과 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN에서 프로비저닝 네트워크용 NIC 1개
 - 이중 NIC 구성 - 프로비저닝 네트워크용 NIC 1개와 외부 네트워크용 다른 NIC
 - 이중 NIC 구성 - 기본 VLAN에서 프로비저닝 네트워크용 NIC 1개와 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN용 다른 NIC
 - 여러 NIC 구성 - 각 NIC에서 다른 오버클라우드 네트워크 유형에 서브넷 사용
- 추가 물리적 NIC는 별도의 네트워크 분리, 연결된 인터페이스 생성 또는 태그된 VLAN 트래픽 위임에 사용할 수 있습니다.
- VLAN을 사용하여 네트워크 트래픽 유형을 분리하는 경우 802.1Q 표준을 지원하는 스위치를 사용하여 태그된 VLAN을 제공하십시오.
- 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 주 NIC를 사용하고, OpenStack 서비스에는 보조 NIC를 사용하십시오.
- 프로비저닝 네트워크 NIC가 director 머신에서 원격 연결에 사용되는 NIC와 같지 않도록 하십시오. director 설치 시 프로비저닝 NIC를 사용하여 브리지가 생성되며, 이로 인해 원격 연결이 모두 끊깁니다. director 시스템에 대한 원격 연결에는 외부 NIC를 사용하십시오.
- 프로비저닝 네트워크에는 현재 환경 크기에 맞는 IP 범위가 필요합니다. 다음 지침을 사용하여 이 범위에 포함할 총 IP 주소 수를 결정하십시오.
 - 프로비저닝 네트워크에 연결된 노드 당 최소 한 개의 IP 주소를 포함합니다.
 - 고가용성 구성을 플래닝하는 경우 클러스터의 가상 IP에 대한 추가 IP 주소를 포함합니다.
 - 환경을 확장할 범위 내에 추가 IP 주소를 포함합니다.



참고

프로비저닝 네트워크에서 IP 주소를 중복으로 사용하지 마십시오. 자세한 내용은 [3.2절. “네트워크 플래닝”](#)을 참조하십시오.



참고

스토리지, 공급자 및 테넌트 네트워크 등에 대한 IP 주소 사용 범위 플래닝에 대한 자세한 내용은 [네트워킹 가이드](#)를 참조하십시오.

- 프로비저닝 NIC로 PXE를 부팅하도록 모든 오버클라우드 시스템을 설정하고, 외부 NIC(및 시스템의 다른 모든 NIC)에서 PXE 부팅을 비활성화하십시오. 또한 프로비저닝 NIC에 PXE 부팅이 하드 디스크 및 CD/DVD 드라이브보다 우선하도록 부팅 순서에서 최상위 위치로 지정합니다.
- 모든 오버클라우드 베어 메탈 시스템에는 IPMI(Intelligent Platform Management Interface)와 같은 지원되는 전원 관리 인터페이스가 필요합니다. 이를 통해 director에서 각 노드의 전원 관리를 제어할 수 있습니다.

- 각 오버클라우드 시스템에 대해 프로비저닝 NIC의 MAC 주소, IPMI NIC의 IP 주소, IPMI 사용자 이름 및 IPMI 비밀번호를 기록해 두십시오. 이 정보는 나중에 오버클라우드 노드를 설정할 때 유용합니다.
- 외부 인터넷에서 인스턴스에 액세스할 필요가 있는 경우 공용 네트워크에서 유동 IP 주소를 할당하고 이 주소를 인스턴스와 연결할 수 있습니다. 인스턴스는 해당 개인 IP를 보유하고 있지만, 네트워크 트래픽에서 NAT를 사용하여 유동 IP 주소를 통과합니다. 유동 IP 주소는 여러 개인 IP 주소가 아니라 단일 인스턴스에만 할당할 수 있습니다. 하지만 유동 IP 주소는 단일 테넌트에서만 사용하도록 지정되어 있으므로 테넌트가 필요에 따라 특정 인스턴스와 연결하거나 연결을 해제할 수 있습니다. 이 구성을 사용하면 해당 인프라가 외부 인터넷에 노출되므로 적합한 보안 관행을 준수하고 있는지 확인해야 합니다.
- 하나의 브리지는 단일 인터페이스 또는 단일 본딩만을 멤버로 하면 Open vSwitch에서 네트워크 루프 발생 위험을 완화할 수 있습니다. 여러 개의 본딩이나 인터페이스가 필요한 경우 여러 브리지를 구성할 수 있습니다.

중요

OpenStack Platform 구현의 보안은 네트워크 환경의 보안에 따라 좌우됩니다. 네트워킹 환경에서 적합한 보안 원칙에 따라 네트워크 액세스가 적절히 제어되는지 확인하십시오. 예를 들면 다음과 같습니다.

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

시스템 보안에 대한 자세한 내용은 다음 문서를 참조하십시오.

- [Red Hat Enterprise Linux 7 보안 가이드](#)
- [Red Hat Enterprise Linux 7 SELinux 사용자 및 관리자 가이드](#)

2.4. 오버클라우드 요구 사항

다음 섹션에서는 오버클라우드 설치 시 개별 시스템과 노드에 대한 요구 사항을 자세히 설명합니다.

참고

SAN(FC-AL, FCoE, iSCSI)에서 오버클라우드 노드를 부팅하는 기능은 아직 지원되지 않습니다.

2.4.1. Compute 노드 요구 사항

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

프로세서

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

메모리

최소 **6GB RAM**. 가상 머신 인스턴스에서 사용하려는 메모리 크기에 따라 이 요구 사항에 **RAM**을 추가합니다.

디스크 공간

최소 **40GB**의 사용 가능한 디스크 공간.

네트워크 인터페이스 카드

최소 **1GB**의 네트워크 인터페이스 카드 한 개. 프로덕션 환경에서는 적어도 두 개 이상의 **NIC**를 사용하는 것이 좋습니다. 연결된 인터페이스에 사용하거나 태그된 **VLAN** 트래픽을 위임하기 위해서는 추가 네트워크 인터페이스를 사용하십시오.

전원 관리

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

2.4.2. Controller 노드 요구 사항

Controller 노드는 **RHEL OpenStack Platform** 환경에서 **Horizon** 대시보드, 백엔드 데이터베이스 서버, **Keystone** 인증 및 고가용성 서비스와 같은 핵심 서비스를 호스팅합니다.

프로세서

Intel 64 또는 **AMD64 CPU** 확장을 지원하는 **64비트 x86** 프로세서.

메모리

최소 메모리 용량은 **32GB**입니다. 하지만 권장 메모리 용량은 **vCPU 수**(**CPU** 코어 수와 하이퍼 스레딩 값을 곱한 값)에 따라 다릅니다. 다음 계산을 참고하십시오.

- **Controller의 최소 RAM 계산:**

- **vCPU**마다 **1.5GB** 메모리를 사용합니다. 예를 들어 **48개**의 **vCPU**가 있는 머신에는 **72GB**의 **RAM**이 있어야 합니다.

- **Controller의 권장 RAM 계산:**

- **vCPU**마다 **3GB** 메모리를 사용합니다. 예를 들어 **48개**의 **vCPU**가 있는 머신에는 **144GB** **RAM**이 있어야 합니다.

메모리 요구 사항 측정에 대한 자세한 내용은 **Red Hat** 고객 포털에서 ["고가용성 컨트롤러에 대한 Red Hat OpenStack Platform 하드웨어 요구 사항"](#)을 참조하십시오.

디스크 스토리지 및 레이아웃

기본적으로 **Telemetry(gnocchi)** 및 **Object Storage(swift)** 서비스는 모두 **Controller**에 설치되어 **root** 디스크를 사용하도록 설정됩니다. 이러한 기본값은 상용 하드웨어에 내장되는 소형 오버클라우드 배포에 적합합니다. 이는 개념 검증 및 테스트의 표준 환경입니다. 이러한 기본값을 사용하면 워크로드 용량 및 성능 측면에서는 떨어지지만 최소의 플래닝으로 오버클라우드 배포가 가능합니다.

하지만 엔터프라이즈 환경에서는 **Telemetry**가 스토리지에 지속적으로 액세스하므로 이 경우 심각한 성능 장애가 발생할 수 있습니다. 그러면 디스크 **I/O**가 과도하게 사용되고 다른 모든 **Controller** 서비스의 성능에 심각한 영향을 미칩니다. 이러한 유형의 환경에서는 오버클라우드를 계획하고 적절하게 구성해야 합니다.

Red Hat은 **Telemetry**와 **Object Storage**에 대한 여러 구성 권장 사항을 제공합니다. 자세한 내용은 [특정 Red Hat OpenStack Platform 서비스에 대한 배포 권장 사항](#)을 참조하십시오.

네트워크 인터페이스 카드

최소 2개의 1GB의 네트워크 인터페이스 카드. 연결된 인터페이스에 사용하거나 태그된 VLAN 트래픽을 위임하기 위해서는 추가 네트워크 인터페이스 카드를 사용하십시오.

전원 관리

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

2.4.3. Ceph Storage 노드 요구 사항

Ceph Storage 노드는 RHEL OpenStack Platform 환경에 스토리지 오브젝트를 제공합니다.

프로세서

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

메모리

메모리 요구 사항은 스토리지 공간에 따라 다릅니다. 이상적인 크기는 1TB 하드 디스크 공간마다 최소 1GB의 메모리를 사용하는 것입니다.

디스크 공간

스토리지 요구 사항은 메모리 크기에 따라 다릅니다. 1TB의 하드 디스크 공간마다 최소 1GB의 메모리를 사용하는 것이 이상적입니다.

디스크 레이아웃

Red Hat Ceph Storage 노드의 권장 설정은 적어도 세 개 이상의 디스크를 다음과 같은 레이아웃으로 구성해야 합니다.

- `/dev/sda` - root 디스크. director가 기본 오버클라우드 이미지를 디스크에 복사합니다.
- `/dev/sdb` - 저널 디스크. 이 디스크는 Ceph OSD 저널용 파티션으로 나뉩니다(예: `/dev/sdb1`, `/dev/sdb2`, `/dev/sdb3` 등). 저널 디스크는 일반적으로 시스템 성능을 지원하기 위한 SSD(솔리드 스테이트 드라이브)입니다.
- `/dev/sdc` 이후 - OSD 디스크. 스토리지 요구 사항에 따라 필요한 만큼 디스크를 사용하십시오.

네트워크 인터페이스 카드

최소 1개의 1GB 네트워크 인터페이스 카드. 프로덕션 환경에서는 적어도 두 개 이상의 NIC를 사용하는 것이 좋습니다. 연결된 인터페이스에 사용하거나 태그된 VLAN 트래픽을 위임하기 위해서는 추가 네트워크 인터페이스 카드를 사용하십시오. 대량의 트래픽에 서비스를 제공하는 OpenStack Platform 환경을 구축하는 경우 스토리지 노드에 10GB 인터페이스를 사용하는 것이 좋습니다.

전원 관리

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

Ceph Storage 클러스터를 사용한 오버클라우드 설치에 대한 자세한 내용은 [오버클라우드용 Red Hat Ceph Storage](#) 가이드를 참조하십시오.

2.4.4. Object Storage 노드 요구 사항

Object Storage 노드는 오버클라우드에 오브젝트 스토리지 계층을 제공합니다. Object Storage 프로키는 Controller 노드에 설치됩니다. 스토리지 계층에는 노드마다 여러 개의 디스크가 있는 베어 메탈 노드가 필요합니다.

프로세서

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

메모리

메모리 요구 사항은 스토리지 공간의 크기에 따라 다릅니다. 가장 좋은 방법은 1TB의 하드 디스크 공간마다 최소 1GB의 메모리를 사용하는 것입니다. 최적의 성능을 위해 특히 워크로드가 작은 파일(100GB 미만)의 경우에는 1TB의 하드 디스크 공간마다 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 다릅니다. 주로 작은 오브젝트를 저장하는 경우 더 많은 SSD 공간을 사용하고, 큰 오브젝트(비디오, 백업)의 경우에는 더 적은 SSD 공간을 사용하십시오.

디스크 레이아웃

권장 노드 구성에는 다음과 비슷한 디스크 레이아웃이 필요합니다.

- **/dev/sda** - root 디스크. **director**가 주요 오버클라우드 이미지를 디스크에 복사합니다.
- **/dev/sdb** - 계정 데이터에 사용됩니다.
- **/dev/sdc** - 컨테이너 데이터에 사용됩니다.
- **/dev/sdd** 이후 - 오브젝트 서버 디스크. 스토리지 요구 사항에 따라 필요한 디스크 수를 사용하십시오.

네트워크 인터페이스 카드

최소 2개의 1GB의 네트워크 인터페이스 카드. 연결된 인터페이스에 사용하거나 태그된 VLAN 트래픽을 위임하기 위해서는 추가 네트워크 인터페이스 카드를 사용하십시오.

전원 관리

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

2.5. 리포지토리 요구 사항

언더클라우드와 오버클라우드 모두 Red Hat Content Delivery Network를 통해 또는 Red Hat Satellite 5 또는 6를 통해 Red Hat 리포지토리에 액세스해야 합니다. Red Hat Satellite 서버를 사용하는 경우 필수 리포지토리를 해당 OpenStack Platform 환경에 동기화합니다. 다음 CDN 채널 이름 목록을 가이드로 사용하십시오.



주의

Open vSwitch(OVS) 2.4.0에서 OVS 2.5.0으로 업그레이드하지 않은 경우 Red Hat Enterprise Linux 7.3 커널로 업그레이드하지 마십시오. 커널만 업그레이드하는 경우 OVS의 작동이 중지됩니다.

표 2.1. OpenStack Platform 리포지토리

이름	리포지토리	요구 사항 설명
----	-------	----------

Red Hat Enterprise Linux 7 Server(RPMs)	rhel-7-server-rpms	기본 운영 체제 리포지토리입니다.
Red Hat Enterprise Linux 7 Server - Extras(RPMs)	rhel-7-server-extras-rpms	Red Hat OpenStack Platform 종속성을 포함합니다.
Red Hat Enterprise Linux 7 Server - RH Common(RPMs)	rhel-7-server-rh-common-rpms	Red Hat OpenStack Platform 배포 및 구성을 위한 툴을 포함합니다.
Red Hat Satellite Tools for RHEL 7 Server RPMs x86_64	rhel-7-server-satellite-tools-6.2-rpms	Red Hat Satellite 6 호스트를 관리하는 툴입니다.
Red Hat Enterprise Linux High Availability (for RHEL 7 Server) (RPMs)	rhel-ha-for-rhel-7-server-rpms	Red Hat Enterprise Linux용고가용성 툴입니다. Controller 노드고가용성을 위해 사용됩니다.
Red Hat Enterprise Linux OpenStack Platform 11 for RHEL 7(RPMs)	rhel-7-server-openstack-11-rpms	Red Hat OpenStack Platform 핵심 리포지토리입니다. 또한 Red Hat OpenStack Platform director용 패키지를 포함합니다.
Red Hat Ceph Storage OSD 2 for Red Hat Enterprise Linux 7 Server(RPMs)	rhel-7-server-rhceph-2-osd-rpms	(Ceph Storage 노드용) Ceph Storage Object Storage 데몬용 리포지토리입니다. Ceph Storage 노드에 설치됩니다.
Red Hat Ceph Storage MON 2 for Red Hat Enterprise Linux 7 Server(RPMs)	rhel-7-server-rhceph-2-mon-rpms	(Ceph Storage 노드용) Ceph Storage 모니터 데몬용 리포지토리입니다. Ceph Storage 노드를 사용하는 OpenStack 환경의 Controller 노드에 설치됩니다.
Red Hat Ceph Storage Tools 2 for Red Hat Enterprise Linux 7 Workstation(RPMs)	rhel-7-server-rhceph-2-tools-rpms	노드가 Ceph Storage 클러스터와 통신할 수 있는 툴을 제공합니다. 오버클라우드를 Ceph Storage 클러스터와 함께 배포하는 경우 모든 노드에 대해 이 리포지토리를 활성화해야 합니다.



참고

오프라인 네트워크의 Red Hat OpenStack Platform 환경에 대한 리포지토리를 구성하려면 Red Hat 고객 포털에서 ["오프라인 환경에서 Red Hat OpenStack Platform Director 구성"](#)을 참조하십시오.

3장. 오버클라우드 플래닝

다음 섹션에서는 Red Hat OpenStack Platform 환경의 여러 측면을 플래닝하는 몇 가지 지침을 제공합니다. 여기에는 노드 역할 정의, 네트워크 토폴로지 플래닝 및 스토리지가 포함됩니다.

3.1. 노드 배포 역할 플래닝

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

Controller

환경을 제어하기 위한 주요 서비스를 제공합니다. 대시보드(**horizon**), 인증(**keystone**), 이미지 스토리지(**glance**), 네트워킹(**neutron**), 오케스트레이션(**heat**) 및 고가용성 서비스(**Controller** 노드를 두 개 이상 사용하는 경우)가 여기에 포함됩니다. Red Hat OpenStack Platform 환경에는 다음 중 하나가 필요합니다.

- 기본 환경을 위한 노드 한 개
- 고가용성 환경을 위한 노드 세 개

노드가 두 개 있거나 4개 이상 있는 환경은 지원되지 않습니다.

Compute

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

Ceph-Storage

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

Cinder-Storage

OpenStack의 Cinder 서비스에 외부 블록 스토리지를 제공하는 호스트입니다. 이 배포 역할은 선택 사항입니다.

Swift-Storage

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

다음 표에서는 다른 오버클라우드 구성 예와 각 시나리오에 사용되는 노드 유형을 정의합니다.

표 3.1. 시나리오에 사용되는 노드 배포 역할

	Controller	Compute	Ceph-Storage	Swift-Storage	Cinder-Storage	합계
소규모 오버클라우드	1	1	-	-	-	2
중간 규모 오버클라우드	1	3	-	-	-	4

추가 Object Storage 및 Block Storage가 있는 중간 규모의 오버클라우드	1	3	-	1	1	6
고가용성이 있는 중간 규모의 오버클라우드	3	3	-	-	-	6
고가용성 및 Ceph Storage가 있는 중간 규모의 오버클라우드	3	3	3	-	-	9

또한 개별 서비스를 사용자 정의 역할로 분할할 것인지 고려하십시오. 작성 가능한 역할 아키텍처에 대한 자세한 내용은 *오버클라우드 고급 사용자 정의 가이드*에서 [8장. 작성 가능한 역할 및 서비스](#)를 참조하십시오.

3.2. 네트워크 플래닝

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

Red Hat OpenStack Platform은 환경의 여러 서브넷에 할당된 별도의 네트워크 트래픽 유형에 다양한 서비스를 매핑합니다. 이러한 네트워크 트래픽 유형은 다음과 같습니다.

표 3.2. 네트워크 유형 할당

네트워크 유형	설명	사용 용도
IPMI	노드의 전원 관리에 사용되는 네트워크입니다. 이 네트워크는 언더클라우드 설치 전에 사전 정의됩니다.	모든 노드
프로비저닝/컨트롤 플레인	director 는 이 네트워크 트래픽 유형을 사용하여 PXE 부팅 시 새 노드를 배포하고, 오버클라우드 베어 메탈 서버에 OpenStack Platform 설치를 오케스트레이션합니다. 이 네트워크는 언더클라우드 설치 전에 사전 정의됩니다.	모든 노드

내부 API	내부 API 네트워크는 API 통신, RPC 메시지 및 데이터베이스 통신을 사용하는 OpenStack 서비스 간 통신에 사용됩니다.	Controller, Compute, Cinder Storage, Swift Storage
테넌트	Neutron은 VLAN 세그멘테이션(각 테넌트 네트워크가 네트워크 VLAN임) 또는 터널링(VXLAN 또는 GRE를 통해)을 사용하여 각 테넌트에 고유한 네트워크를 제공합니다. 네트워크 트래픽은 각 테넌트 네트워크 내에서 분할됩니다. 각 테넌트 네트워크에는 이와 연결된 IP 서브넷이 있으며, 네트워크 네임스페이스로 인해 여러 테넌트 네트워크에서 충돌을 일으키지 않고 동일한 주소 범위를 사용할 수 있습니다.	Controller, Compute
Storage	Block Storage, NFS, iSCSI 등입니다. 성능상의 이유로 완전히 별도의 스위치 패브릭으로 분리하는 것이 좋습니다.	모든 노드
스토리지 관리	OpenStack Object Storage(swift)는 이 네트워크를 사용하여 참여하는 복제 노드 간에 데이터 오브젝트를 동기화합니다. 프록시 서비스는 사용자 요청과 기존 스토리지 계층 사이의 중간 인터페이스 역할을 합니다. 프록시는 들어오는 요청을 수신하고 필요한 복제본을 찾아서 요청된 데이터를 검색합니다. Ceph 백엔드를 사용하는 서비스는 Ceph와 직접 상호 작용하지 않지만, 프론트엔드 서비스를 사용하므로 스토리지 관리 네트워크를 통해 연결됩니다. RBD 드라이버는 예외로, 이 트래픽은 Ceph에 직접 연결됩니다.	Controller, Ceph Storage, Cinder Storage, Swift Storage
외부 네트워크	OpenStack 대시보드(horizon)를 실행하여 그래픽 시스템 관리, OpenStack 서비스 용 공용 API를 호스팅하는 인스턴스로 들어오는 네트워크 트래픽의 SNAT 처리를 수행합니다. 외부 네트워크가 개인 IP 주소를 사용하는 경우(RFC-1918에 따라) 인터넷에서 들어오는 트래픽에 대해 NAT를 수행해야 합니다.	Controller

유동 IP	수신 트래픽이 유동 IP 주소와 테넌트 네트워크의 인스턴스에 실제로 할당된 IP 주소 간에 1:1 IP 주소 매핑을 사용하는 인스턴스에 연결할 수 있습니다. 외부 네트워크와는 분리된 VLAN에서 유동 IP를 호스팅하는 경우 유동 IP VLAN을 Controller 노드로 트렁크 연결을 사용하고 오버클라우드 생성 후 Neutron 을 통해 VLAN을 추가할 수 있습니다. 그러면 여러 브리지에 연결된 유동 IP 네트워크를 여러 개 생성할 수 있습니다. VLAN은 트렁크 연결되지만, 인터페이스로 구성되지는 않습니다. 대신 neutron 이 각 유동 IP 네트워크에 대해 선택한 브리지에서 VLAN 세그멘테이션 ID로 OVS 포트를 생성합니다.	Controller
관리	SSH 액세스, DNS 트래픽 및 NTP 트래픽과 같은 시스템 관리 기능을 제공합니다. 또한 이 네트워크는 Controller 노드 이외의 노드에 대해 게이트웨이 역할을 합니다.	모든 노드

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



참고

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

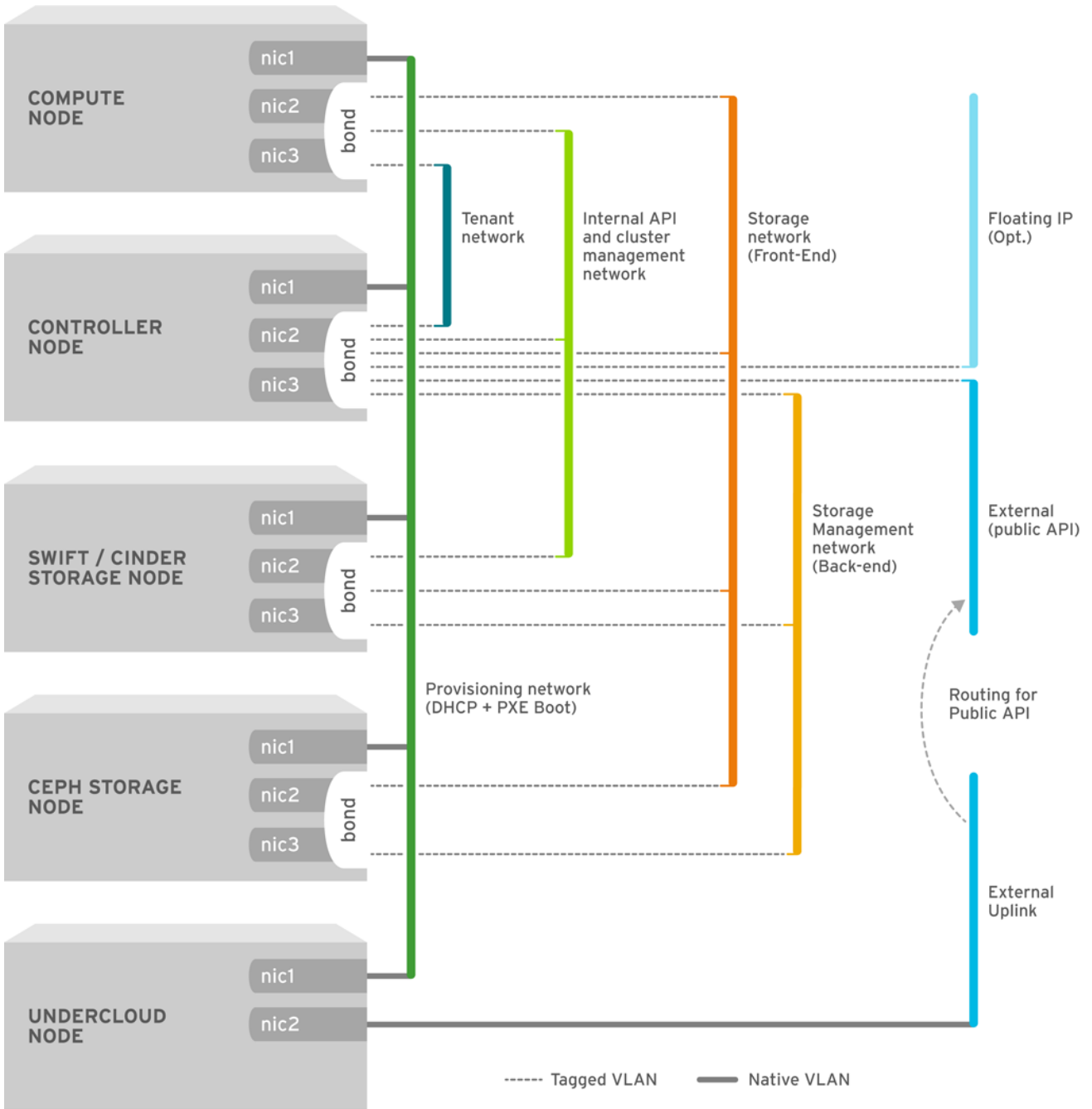
director는 이러한 트래픽 유형 중 6개의 유형을 특정 서브넷이나 VLAN에 매핑하는 방법을 제공합니다. 이러한 트래픽 유형은 다음과 같습니다.

- 내부 API
- Storage
- 스토리지 관리
- 테넌트 네트워크
- 외부 네트워크

• 관리

할당되지 않은 네트워크는 모두 프로비저닝 네트워크와 동일한 서브넷에 자동으로 할당됩니다.

아래 다이어그램은 네트워크가 별도 VLAN으로 분리된 네트워크 토폴로지 예를 보여줍니다. 각 오버클라우드 노드는 본딩에 두 가지 인터페이스(**nic2** 및 **nic3**)를 사용하여 해당 VLAN을 통해 이러한 네트워크를 제공합니다. 또한 각 오버클라우드 노드는 **nic1**을 사용하여 기본 VLAN을 통해 프로비저닝 네트워크로 언더클라우드와 통신합니다.



OPENSTACK_364029_0715

다음 표는 다른 네트워크 레이아웃을 매핑하는 네트워크 트래픽 예를 제공합니다.

표 3.3. 네트워크 매핑

	매핑	총 인터페이스 수	총 VLAN 수
--	----	-----------	----------

외부 액세스 권한이 있는 플랫 네트워크	네트워크 1 - 프로비저닝, 내부 API, 스토리지, 스토 리지 관리, 테넌트 네트워 크 네트워크 2 - 외부, 유동 IP(오버클라우드 생성 후 매핑됨)	2	2
분리된 네트워크	네트워크 1 - 프로비저닝 네트워크 2 - 내부 API 네트워크 3 - 테넌트 네트 워크 네트워크 4 - 스토리지 네트워크 5 - 스토리지 관 리 네트워크 6 - 스토리지 관 리 네트워크 7 - 외부, 유동 IP(오버클라우드 생성 후 매핑됨)	3(본딩된 2개의 인터페이 스 포함)	7

3.3. 스토리지 플래닝

director는 오버클라우드 환경에 여러 스토리지 옵션을 제공합니다. 이러한 옵션은 다음과 같습니다.

Ceph Storage 노드

director가 Red Hat Ceph Storage를 사용하여 확장 가능한 스토리지 노드 집합을 생성합니다. 오버클라우드는 각종 노드를 다음의 목적으로 사용합니다.

- **이미지** - Glance는 VM의 이미지를 관리합니다. 이미지는 변경할 수 없습니다. OpenStack에서는 이미지를 바이너리 Blob으로 처리하고 그에 따라 이미지를 다운로드합니다. glance를 사용하여 이미지를 Ceph 블록 장치에 저장할 수 있습니다.
- **볼륨** - Cinder 볼륨은 블록 장치입니다. OpenStack에서는 볼륨을 사용하여 VM을 부팅하거나, 실행 중인 VM에 볼륨을 연결합니다. OpenStack에서는 Cinder 서비스를 사용하여 볼륨을 관리합니다. Cinder를 사용하면 이미지의 CoW (copy-on-write) 복제본을 사용하여 VM을 부팅할 수 있습니다.
- **게스트 디스크** - 게스트 디스크는 게스트 운영 체제 디스크입니다. 기본적으로 nova로 가상 머신을 부팅하면 해당 디스크가 하이퍼바이저의 파일 시스템에 파일로 표시됩니다(일반적으로 `/var/lib/nova/instances/<uuid>/` 아래). cinder를 사용하지 않고 Ceph 내에서 모든 가상 머신을 직접 부팅할 수 있으며, 라이브 마이그레이션 프로세스를 통해 쉽게 유지보수 작업을 수행할 수 있으므로 유용합니다. 또한 하이퍼바이저가 중지되는 경우 편리하게 nova evacuate를 트리거하고 다른 위치에서 가상 머신을 중단 없이 실행할 수 있습니다.



중요

Ceph에서는 가상 머신 디스크를 호스팅할 QCOW2를 지원하지 않습니다. Ceph에서 가상 머신을 부팅하려면(ephemeral 백엔드 또는 볼륨에서 부팅) glance 이미지 포맷이 **RAW**여야 합니다.

자세한 내용은 [Red Hat Ceph Storage 아키텍처 가이드](#) 를 참조하십시오.

Swift Storage 노드

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

4장. 언더클라우드 설치

Red Hat OpenStack Platform 환경 구축의 첫 단계는 **director**를 언더클라우드 시스템에 설치하는 것입니다. 여기에는 필요한 서브스크립션 및 리포지토리 활성화하기 위해 몇 가지 단계를 수행해야 합니다.

4.1. DIRECTOR 설치 사용자 생성

director 설치 프로세스를 진행하려면 **root**가 아닌 사용자가 명령을 실행해야 합니다. 다음 명령을 사용하여 **stack**이라는 사용자를 생성하고 암호를 설정하십시오.

```
[root@director ~]# useradd stack
[root@director ~]# passwd stack # specify a password
```

sudo 사용 시 이 사용자가 암호를 요구하지 않도록 합니다.

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

새로운 **stack** 사용자로 전환합니다.

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

director 설치를 **stack** 사용자로 계속 진행합니다.

4.2. 템플릿 및 이미지 용 디렉터리 생성

director에서 시스템 이미지와 Heat 템플릿을 사용하여 오버클라우드 환경을 생성합니다. 이러한 파일을 구성된 대로 유지하려면 다음과 같이 이미지 및 템플릿에 디렉터리를 생성하는 것이 좋습니다.

```
$ mkdir ~/images
$ mkdir ~/templates
```

이 가이드의 다른 섹션에서는 이 두 디렉터리를 사용하여 특정 파일을 저장합니다.

4.3. 시스템의 호스트 이름 설정

director에는 설치 및 구성 프로세스에 대해 정규화된 도메인 이름이 필요합니다. 따라서, **director** 호스트의 호스트 이름을 설정해야 하는 경우가 있습니다. 다음 명령으로 호스트의 호스트 이름을 확인하십시오.

```
$ hostname # Checks the base hostname
$ hostname -f # Checks the long hostname (FQDN)
```

필요한 경우 **hostnamectl**을 사용하여 호스트 이름을 설정합니다.

```
$ sudo hostnamectl set-hostname manager.example.com
$ sudo hostnamectl set-hostname --transient manager.example.com
```

director에는 **/etc/hosts**에 시스템의 호스트 이름 및 기본 이름도 입력해야 합니다. 예를 들어, 시스템 이름이 **manager.example.com**으로 지정된 경우 **/etc/hosts**에 다음과 같은 항목이 필요합니다.

```
127.0.0.1    manager.example.com manager localhost localhost.localdomain
localhost4  localhost4.localhost4
```

4.4. 시스템 등록

Red Hat OpenStack Platform director를 설치하려면 먼저, Red Hat Subscription Manager를 사용하여 호스트 시스템을 등록하고 필수 채널에 등록합니다.

1. 고객 포털 사용자 이름과 암호를 입력하라는 메시지가 표시되면 입력하여 콘텐츠 전송 네트워크 (Content Delivery Network)에 시스템을 등록합니다.

```
$ sudo subscription-manager register
```

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

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

3. Pool ID 값을 찾아서 Red Hat OpenStack Platform 11 인타이틀먼트에 연결합니다.

```
$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

4. 기본 리포지토리를 모두 비활성화한 다음 필수 Red Hat Enterprise Linux 리포지토리를 활성화합니다.

```
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos --enable=rhel-7-server-rpms --
enable=rhel-7-server-extras-rpms --enable=rhel-7-server-rh-common-
rpms --enable=rhel-ha-for-rhel-7-server-rpms --enable=rhel-7-server-
openstack-11-rpms
```

이러한 리포지토리에는 director 설치에 필요한 패키지가 들어 있습니다.



중요

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

시스템에서 업데이트를 수행하여 최신 기본 시스템 패키지가 사용되는지 확인합니다.

```
$ sudo yum update -y
$ sudo reboot
```

이제 시스템에서 **director**를 설치할 준비가 되었습니다.

4.5. DIRECTOR 패키지 설치

다음 명령을 사용하여 **director** 설치 및 구성에 필요한 필수 명령줄 툴을 설치합니다.

```
$ sudo yum install -y python-tripleoclient
```

이렇게 하면 **director** 설치에 필요한 모든 패키지가 설치됩니다.

4.6. DIRECTOR 구성

director 설치 프로세스에는 네트워크 구성을 결정하는 특정 설정이 필요합니다. 이 설정은 **stack** 사용자의 홈 디렉터리에 있는 템플릿에 **undercloud.conf**로 저장되어 있습니다.

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

```
$ cp /usr/share/instack-undercloud/undercloud.conf.sample
~/undercloud.conf
```

undercloud.conf 파일에는 언더클라우드를 구성하는 설정이 들어 있습니다. 매개 변수를 생략하거나 주석 처리하는 경우 언더클라우드 설치에 기본값이 사용됩니다.

템플릿에는 **[DEFAULT]** 섹션과 **[auth]** 섹션이 포함되어 있습니다. **[DEFAULT]** 섹션에는 다음 매개 변수가 들어 있습니다.

undercloud_hostname

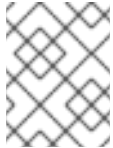
언더클라우드에 대해 정규화된 호스트 이름을 정의합니다. 설정되어 있는 경우 언더클라우드 설치 시 모든 시스템의 호스트 이름이 설정됩니다. 설정되어 있지 않은 경우 언더클라우드에서 현재 호스트 이름을 사용하지만 사용자가 모든 시스템의 호스트 이름을 적절하게 설정해야 합니다.

local_ip

director의 프로비저닝 NIC에 대해 정의된 IP 주소입니다. 이는 **director**에서 해당 DHCP 및 PXE 부팅 서비스에 사용하는 IP 주소이기도 합니다. 프로비저닝 네트워크에 다른 서브넷을 사용하지 않는 경우, 예를 들어 환경에서 기존 IP 주소 또는 서브넷과 충돌하는 경우 이 값을 기본 **192.168.24.1/24**로 남겨 두십시오.

network_gateway

오버클라우드 인스턴스의 게이트웨이로, 트래픽을 외부 네트워크에 전달하는 언더클라우드 호스트입니다. **director**에 다른 IP 주소를 사용하지 않거나 외부 게이트웨이를 직접 사용하지 않을 경우 이 값을 기본 **192.168.24.1**로 남겨 두십시오.



참고

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

undercloud_public_host

SSL/TLS를 사용할 때 **director**의 공용 API에 대해 정의된 IP 주소로, SSL/TLS를 통해 외부 **director** 엔드포인트에 액세스하는 IP 주소입니다. **director** 구성은 이 IP 주소를 해당 소프트웨어 브리지에 /32 넷마스크를 사용하는 라우팅된 IP 주소로 연결합니다.

undercloud_admin_host

SSL/TLS 사용 시 **director**의 관리 API에 대해 정의된 IP 주소로, SSL/TLS로 관리 엔드포인트 액세스를 위한 IP 주소입니다. **director** 설정에 따라 이 IP 주소를 /32 넷마스크를 사용하는 라우팅된 IP 주소로 해당 소프트웨어 브리지에 연결합니다.

undercloud_nameservers

언더클라우드 호스트 이름 확인에 사용할 DNS 이름 서버 목록입니다.

undercloud_ntp_servers

언더클라우드의 날짜 및 시간을 동기화하는데 사용되는 네트워크 시간 프로토콜 서버의 목록입니다.

undercloud_service_certificate

OpenStack SSL/TLS 통신을 위한 인증서 위치 및 파일 이름입니다. 이 인증서를 신뢰할 수 있는 인증 기관에서 가져오는 것이 가장 좋습니다. 그렇지 않은 경우 [부록 A. SSL/TLS 인증서 구성](#)의 지침을 사용하여 자체 서명된 인증서를 생성하십시오. 이러한 지침에는 자체 서명된 인증서이든, 또는 인증 기관에서 서명된 인증서이든 관계없이 인증서의 SELinux 컨텍스트를 설정하는 방법이 포함되어 있습니다.

generate_service_certificate

언더클라우드 설치 중에 **undercloud_service_certificate** 매개 변수에 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** 매개 변수에 정의된 CA는 이 인증서에 서명합니다.

certificate_generation_ca

요청한 인증서에 서명하는 CA의 **certmonger** 닉네임입니다. **generate_service_certificate** 매개 변수를 설정한 경우에만 이 옵션을 사용하십시오. **local** CA를 선택한 경우 **certmonger**에서 로컬 CA 인증서를 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem**에 추출하고 신뢰 체인에 추가합니다.

service_principal

인증서를 사용하는 서비스에 대한 Kerberos 사용자입니다. **FreeIPA**에서 처럼 CA에 Kerberos 사용자가 필요한 경우에만 사용합니다.

local_interface

director의 프로비저닝 NIC 용으로 선택한 인터페이스로, **director**에서 해당 DHCP 및 PXE 부팅 서비스에 사용하는 장치이기도 합니다. 이 값을 선택한 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 예에서 외부 NIC는 **eth0**을 사용하고, 프로비저닝 NIC는 현재 구성되지 않은 **eth1**을 사용합니다. 이 경우에는 **local_interface**를 **eth1**로 설정하십시오. 설정 스크립트는 이 인터페이스를 **inspection_interface** 매개 변수로 정의된 사용자 브리지에 연결합니다.

local_mtu

local_interface에 사용할 MTU입니다.

network_cidr

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

masquerade_network

외부 액세스를 위해 위장할 네트워크를 정의합니다. 그러면 프로비저닝 네트워크에 **NAT**(네트워크 주소 변환)가 제공되어 **director**를 통해 외부 액세스가 가능합니다. 프로비저닝 네트워크에 다른 서브넷을 사용하지 않는 경우 이 값을 기본값(**192.168.24.0/24**)으로 두십시오.

dhcp_start; dhcp_end

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

hieradata_override

hieradata 덮어쓰기 파일에 대한 경로입니다. 설정한 경우, 언더클라우드 설치 시 이 파일이 **/etc/puppet/hieradata**에 복사되고 계층에서 첫 번째 파일로 설정됩니다. 사용자 정의 구성을 **undercloud.conf** 매개 변수 외의 서비스에 제공할 때 사용합니다.

net_config_override

네트워크 구성 덮어쓰기 템플릿에 대한 경로입니다. 설정한 경우, 언더클라우드에서 **JSON** 포맷의 템플릿을 사용하여 네트워킹을 **os-net-config**로 구성합니다. 이는 **undercloud.conf**에 설정된 네트워크 매개 변수를 무시합니다. **/usr/share/instack-undercloud/templates/net-config.json.template**에서 예를 참조하십시오.

inspection_interface

director에서 노드 **introspection**에 사용하는 브리지입니다. 이 브리지는 **director** 구성으로 생성되는 사용자 정의 브리지입니다. **LOCAL_INTERFACE**가 이 브리지에 연결됩니다. 이 브리지를 기본 **br-ctlplane**으로 두십시오.

inspection_iprange

director의 **introspection** 서비스에서 **PXE** 부팅 및 프로비저닝 프로세스 중에 사용하는 **IP** 주소의 범위입니다. 콤마로 구분된 값을 사용하여 이 범위의 시작과 끝을 정의합니다. 예를 들면 **192.168.24.100, 192.168.24.120**과 같습니다. 이 범위에 노드에 대한 충분한 **IP** 주소가 포함되어 있고, **dhcp_start** 및 **dhcp_end**의 범위와 충돌하지 않는지 확인하십시오.

inspection_extras

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

inspection_runbench

노드 **introspection** 중에 벤치마크 집합을 실행합니다. 활성화하려면 **true**로 설정합니다. 등록된 노드의 하드웨어를 검사할 때 벤치마크 분석을 수행하려는 경우 이 옵션이 필요합니다. 자세한 내용은 [5.2 절. “노드의 하드웨어 검사”](#)를 참조하십시오.

inspection_enable_uefi

UEFI 전용 펌웨어를 사용하여 노드의 introspection 지원 여부를 정의합니다. 자세한 내용은 [부록 D. 대체 부팅 모드](#)를 참조하십시오.

enable_node_discovery

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

discovery_default_driver

자동으로 등록된 노드의 기본 드라이버를 설정합니다. **enable_node_discovery**를 활성화해야 하며, 드라이버를 **enabled_drivers** 목록에 포함해야 합니다. 지원되는 드라이버 목록에 대해서는 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

undercloud_debug

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

undercloud_update_packages

언더클라우드 설치 중 패키지 업데이트 여부를 정의합니다.

enable_tempest

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

enable_telemetry

언더클라우드에 OpenStack Telemetry 서비스(ceilometer, aodh, panko, gnocchi) 설치 여부를 정의합니다. Red Hat OpenStack Platform 11에서 Telemetry 메트릭스 백엔드는 gnocchi에서 제공합니다. **enable_telemetry** 매개 변수를 **true**로 설정하면 Telemetry 서비스가 자동으로 설치 및 설정됩니다. 기본값은 **false**로 언더클라우드에서 Telemetry 서비스가 비활성화됩니다.

enable_ui

director의 웹 UI 설치 여부를 정의합니다. 이 경우 그래픽 웹 인터페이스를 통해 오버클라우드 플래닝 및 배포를 수행할 수 있습니다. 자세한 내용은 [6장. 웹 UI로 기본 오버클라우드 설정](#)을 참조하십시오. UI는 **undercloud_service_certificate** 또는 **generate_service_certificate**를 사용하여 활성화된 SSL/TLS에서만 사용할 수 있습니다.

enable_validations

검증을 실행하기 위해 필요한 항목을 설치할지 여부를 정의합니다.

enable_legacy_ceilometer_api

언더클라우드에서 기존 OpenStack Telemetry 서비스(Ceilometer) API 활성화 여부를 정의합니다. 기존 API가 사용되지 않고 이후 릴리스에서 제거됩니다. **enable_telemetry**와 함께 설치된 최신 구성 요소를 사용하십시오.

enable_novajoin

언더클라우드에서 novajoin 메타데이터 서비스 설치 여부를 정의합니다.

ipa_otp

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

ipxe_enabled

iPXE 또는 표준 PXE 사용 여부를 정의합니다. 기본값은 **true**이며, iPXE를 활성화합니다. 표준 PXE로 설정하려면 **false**로 설정하십시오. 자세한 내용은 [부록 D. 대체 부팅 모드](#)를 참조하십시오.

scheduler_max_attempts

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

clean_nodes

배포 사이에 그리고 **introspection** 후에 하드 드라이브를 초기화할 것인지 여부를 정의합니다. 지우기를 활성화하면 노드 등록 및 인스턴스 삭제 시간이 늘어나며, 삭제 후에는 오버클라우드를 복원할 수 없습니다.

enabled_drivers

언더클라우드에 대해 활성화할 베어 메탈 드라이버 목록입니다. 지원되는 드라이버 목록은 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

[auth] 섹션에는 다음 매개 변수가 들어 있습니다.

undercloud_db_password; undercloud_admin_token; undercloud_admin_password; undercloud_glance_password; etc

나머지 매개 변수는 모든 **director** 서비스에 대한 액세스 정보를 정의하는데 사용됩니다. 이 값을 변경할 필요가 없습니다. **undercloud.conf**에 비어 있는 경우 **director**의 구성 스크립트는 이러한 값을 자동으로 생성합니다. 구성 스크립트가 완료되면 모든 값을 검색할 수 있습니다.



중요

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

네트워크에 따라 이러한 매개 변수에 대한 값을 변경하십시오. 완료되면 파일을 저장하고 다음 명령을 실행하십시오.

```
$ openstack undercloud install
```

이 명령은 **director**의 구성 스크립트를 실행합니다. **director**가 추가 패키지를 설치하고, **undercloud.conf**의 설정에 맞게 해당 서비스를 구성합니다. 이 스크립트는 완료하는 데 몇 분이 걸립니다.

구성 스크립트는 완료 시 다음 두 파일을 생성합니다.

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

이 구성은 모든 **OpenStack Platform** 서비스를 자동으로 시작합니다. 다음 명령을 사용하여 활성화된 서비스를 확인하십시오.

```
$ sudo systemctl list-units openstack-*
```

stack 사용자를 초기화하여 명령줄 툴을 사용하려면 다음 명령을 실행하십시오.

```
$ source ~/stackrc
```

이제 **director**의 명령줄 툴을 사용할 수 있습니다.

4.7. 오버클라우드 노드의 이미지 가져오기

director에는 오버클라우드 노드를 프로비저닝할 다음과 같은 여러 디스크 이미지가 필요합니다.

- **introspection** 커널 및 램디스크 - PXE 부팅을 통한 베어 메탈 시스템 **introspection**에 사용됩니다.

- 배포 커널 및 램디스크 - 시스템 프로비저닝 및 배포에 사용됩니다.
- 오버클라우드 커널, 램디스크 및 전체 이미지 - 노드의 하드 디스크에 기록된 기본 오버클라우드 시스템입니다.

이러한 이미지는 **rhosp-director-images** 및 **rhosp-director-images-ipa** 패키지에서 가져옵니다.

```
$ sudo yum install rhosp-director-images rhosp-director-images-ipa
```

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

```
$ cd ~/images
$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-11.0.tar
/usr/share/rhosp-director-images/ironic-python-agent-latest-11.0.tar; do
tar -xvf $i; done
```

이러한 이미지를 **director**로 가져옵니다.

```
$ openstack overcloud image upload --image-path /home/stack/images/
```

이렇게 하면 **bm-deploy-kernel**, **bm-deploy-ramdisk**, **overcloud-full**, **overcloud-full-initrd**, **overcloud-full-vmlinuz** 이미지가 **director**에 업로드됩니다. 이러한 이미지는 배포 및 오버클라우드의 이미지입니다. 스크립트는 **director**의 PXE 서버에 **introspection** 이미지도 설치합니다.

CLI에서 이미지 목록을 확인합니다.

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

이 목록에는 **introspection** PXE 이미지가 표시되지 않습니다. **director**에서는 이러한 파일을 **/httpboot**에 복사합니다.

```
[stack@host1 ~]$ ls -l /httpboot
total 341460
-rwxr-xr-x. 1 root          root          5153184 Mar 31 06:58
agent.kernel
-rw-r--r--. 1 root          root          344491465 Mar 31 06:59
agent.ramdisk
-rw-r--r--. 1 ironic-inspector ironic-inspector 337 Mar 31 06:23
inspector.ipxe
```



참고

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

4.8. 언더클라우드의 NEUTRON 서브넷에서 네임 서버 설정

오버클라우드 노드에는 DNS를 통해 호스트 이름을 확인할 수 있도록 네임 서버가 필요합니다. 네트워크 분리가 없는 표준 오버클라우드의 경우 언더클라우드의 **neutron** 서브넷을 사용하여 네임 서버가 정의됩니다. 환경의 네임 서버를 정의하려면 다음 명령을 사용하십시오.

```
$ openstack subnet list
$ openstack subnet set --dns-nameserver [nameserver1-ip] --dns-nameserver [nameserver2-ip] [subnet-uuid]
```

서브넷을 표시하여 네임 서버를 확인합니다.

```
$ openstack subnet show [subnet-uuid]
+-----+-----+
| Field          | Value                                |
+-----+-----+
| ...            |                                     |
| dns_nameservers | 8.8.8.8                             |
| ...            |                                     |
+-----+-----+
```



중요

서비스 트래픽을 별도의 네트워크에 분리하려는 경우 오버클라우드 노드에서 네트워크 환경 파일에 **DnsServer** 매개 변수를 사용합니다.

4.9. 언더클라우드 백업

Red Hat은 언더클라우드 호스트와 Red Hat OpenStack Platform director에서 중요한 데이터를 백업하는 프로세스를 제공합니다. 언더클라우드 백업에 대한 자세한 내용은 "[Director 언더클라우드 백업 및 복원](#)" 가이드를 참조하십시오.

4.10. 언더클라우드 설정 완료

이렇게 하면 언더클라우드 설정이 완료됩니다. 다음 장에서는 노드 등록, 검사 및 여러 노드 역할 태그 지정을 비롯하여 오버클라우드의 기본적인 설정에 대해 설명합니다.

5장. CLI 툴로 기본 오버클라우드 설정

이 장에서는 CLI 툴을 사용하는 OpenStack Platform 환경에 대한 기본적인 설정 단계를 설명합니다. 기본 설정을 사용하는 오버클라우드에는 사용자 정의 기능이 포함되어 있지 않지만, 고급 옵션을 이 기본 오버클라우드에 추가하고 [고급 오버클라우드 사용자 정의](#) 가이드의 지침을 사용하여 사양에 맞게 사용자 정의를 할 수 있습니다.

이 장의 예에서는 모든 노드가 전원 관리에 IPMI를 사용하는 베어 메탈 시스템으로 되어 있습니다. 지원되는 전원 관리 유형 및 옵션에 대한 자세한 내용은 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

워크플로우

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

요구 사항

- [4장. 언더클라우드 설치](#)에서 생성한 **director** 노드
- 노드에 사용하는 베어 메탈 머신 세트. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다(오버클라우드 역할에 대한 자세한 내용은 [3.1절. “노드 배포 역할 플래닝”](#) 참조). 이러한 머신은 각 노드 유형에 대해 설정된 요구 사항을 준수해야 합니다. 이러한 요구 사항은 [2.4절. “오버클라우드 요구 사항”](#)을 참조하십시오. 이러한 노드에는 운영 체제가 필요하지 않습니다. **director**에서 Red Hat Enterprise Linux 7 이미지를 각 노드에 복사합니다.
- 기본 VLAN으로 구성된 프로비저닝 네트워크에 대한 한 개의 네트워크 연결. 모든 노드는 이 네트워크에 연결해야 하며, [2.3절. “네트워킹 요구 사항”](#)에 설정된 요구 사항을 준수해야 합니다. 이 장의 예에서는 192.168.24.0/24를 다음 IP 주소가 할당된 프로비저닝 서브넷으로 사용합니다.

표 5.1. 프로비저닝 네트워크 IP 할당

노드 이름	IP 주소	MAC 주소	IPMI IP 주소
Director	192.168.24.1	aa:aa:aa:aa:aa:aa	필요 없음
Controller	DHCP 정의됨	bb:bb:bb:bb:bb:bb	192.168.24.205
Compute	DHCP 정의됨	cc:cc:cc:cc:cc:cc	192.168.24.206

- 다른 모든 네트워크 유형은 OpenStack 서비스에 프로비저닝 네트워크를 사용하지만 다른 네트워크 트래픽 유형에 대해 추가로 네트워크를 생성할 수 있습니다.

5.1. 오버클라우드에 노드 등록

director에는 수동으로 만든 노드 정의 템플릿이 필요합니다. 이 파일(**instackenv.json**)은 JSON 포맷 파일을 사용하며, 노드의 하드웨어 및 전원 관리 정보가 포함되어 있습니다. 예를 들면 노드 두 개를 등록할 템플릿은 다음과 같이 표시될 수 있습니다.


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

이 템플릿은 다음 속성을 사용합니다.

pm_type

사용할 전원 관리 드라이버입니다. 이 예에서는 IPMI 드라이버(**pxe_ipmitool**)를 사용합니다.

pm_user; pm_password

IPMI 사용자 이름 및 암호입니다.

pm_addr

IPMI 장치의 IP 주소입니다.

mac

(선택 사항) 노드에 있는 네트워크 인터페이스의 **MAC** 주소 목록입니다. 각 시스템의 프로비저닝 **NIC**에는 **MAC** 주소만 사용하십시오.

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.



참고

지원되는 전원 관리 유형 및 해당 옵션에 대해서는 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

템플릿을 생성한 후 파일을 **stack** 사용자의 홈 디렉터리(`/home/stack/instackenv.json`)에 저장한 후 다음 명령을 사용하여 **director**로 가져옵니다.

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

이렇게 하면 템플릿을 가져와서 템플릿의 각 노드가 **director**에 등록됩니다.

노드 등록 및 설정이 완료되면 CLI에서 이러한 노드 목록을 확인합니다.

```
$ openstack baremetal node list
```

5.2. 노드의 하드웨어 검사

director는 각 노드에서 **introspection** 프로세스를 실행할 수 있습니다. 이 프로세스를 실행하면 각 노드가 PXE를 통해 **introspection** 에이전트가 부팅됩니다. 이 에이전트는 노드에서 하드웨어 데이터를 수집하여 **director**에 다시 보냅니다. 그러면 **director**에서 이 **introspection** 데이터를 **director**에서 실행되는 OpenStack Object Storage(swift) 서비스에 저장합니다. **director**는 프로필 태그, 벤치마킹 및 root 디스크 수동 할당과 같은 여러 목적에 하드웨어 정보를 사용합니다.



참고

정책 파일을 생성하여 **introspection** 이후 바로 프로필에 노드를 자동으로 태깅할 수 있습니다. 정책 파일 생성 및 **introspection** 프로세스에 정책 파일을 포함하는 작업에 대한 자세한 내용은 [부록 E. 자동 프로필 태깅](#)을 참조하십시오. 또는 [5.3절. “프로필에 노드 태그”](#)의 지침에 따라 프로파일에 노드를 수동으로 태그할 수 있습니다.

다음 명령을 실행하여 각 노드의 하드웨어 속성을 확인합니다.

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

- **--all-manageable** 옵션은 관리되는 상태의 노드만 검사합니다. 이 예에서는 모든 노드가 해당됩니다.
- **--provide** 옵션은 **introspection** 이후 모든 노드를 **active** 상태로 재설정합니다.

다른 터미널 창에서 다음 명령을 사용하여 **introspection** 진행 상황을 모니터링합니다.

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



중요

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

introspection이 완료되면 모든 노드가 **active** 상태로 변경됩니다.

개별적으로 노드 Introspection 실행

active 노드에서 개별적으로 introspection을 실행하려면 노드를 관리 모드로 설정하고 introspection을 실행합니다.

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

introspection이 완료되면 노드가 **active** 상태로 변경됩니다.

초기 Introspection 이후 노드 Introspection 실행

초기 introspection 이후 모든 노드는 **--provide** 옵션으로 인해 **active** 상태로 전환됩니다. 초기 introspection 이후 모든 노드에서 introspection을 실행하려면 모든 노드를 **manageable** 상태로 설정하고 일괄적으로 introspection 명령을 실행합니다.

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

introspection이 완료되면 모든 노드가 **active** 상태로 변경됩니다.

5.3. 프로필에 노드 태그

각 노드의 하드웨어 등록 및 검사 후 특정 프로필에 노드를 태그합니다. 이러한 프로파일 태그는 노드가 플레이버에 일치된 다음 해당 플레이버가 배포 역할에 할당됩니다. 다음 예는 **Controller** 노드에 대한 역할, 플레이버, 프로파일 및 노드 사이의 관계를 보여줍니다.

유형	설명
역할	Controller 역할은 컨트롤러 노드 설정 방법을 정의합니다.
플레이버	control 플레이버는 노드에서 컨트롤러로 사용할 하드웨어 프로파일을 정의합니다. director 에서 사용할 노드를 결정할 수 있도록 이 플레이버를 Controller 역할에 할당합니다.
프로파일	control 프로파일은 control 플레이버에 적용하는 태그입니다. 이 프로파일은 플레이버에 속한 노드를 정의합니다.
노드	또한 control 프로파일 태그를 개별 노드에 적용하면 control 플레이버에 그룹화되며, 결과적으로 director 에서는 Controller 역할을 사용하여 이러한 개별 노드를 설정합니다.

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



참고

대부분의 노드는 프로파일 자동 태그를 사용합니다. 자세한 내용은 [부록 E. 자동 프로파일 태그](#)를 참조하십시오.

노드를 특정 프로파일에 태그하려면 **profile** 옵션을 각 노드의 **properties/capabilities** 매개 변수에 추가합니다. 예를 들어 노드를 **Controller** 및 **Compute** 프로파일에 각각 태그하려면 다음 명령을 사용합니다.

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

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

이러한 명령은 또한 각 노드 부팅 방법을 정의하는 **boot_option:local** 매개 변수를 설정합니다. 하드웨어에 따라 노드가 기본 BIOS 모드 대신 UEFI를 사용하여 부팅되도록 하기 위해 **boot_mode** 매개 변수를 **uefi**에 추가해야 할 수도 있습니다. 자세한 내용은 [D.2절. “UEFI 부팅 모드”](#)를 참조하십시오.

노드 태그를 완료한 후 할당된 프로파일 또는 가능한 프로파일을 확인합니다.

```
$ openstack overcloud profiles list
```

사용자 정의 역할 프로파일

사용자 정의 역할을 사용하는 경우 이러한 새 역할을 적용할 추가 플레이버 및 프로파일을 생성해야 할 수도 있습니다. 예를 들어 **Networker** 역할의 새 플레이버를 생성하려면 다음 명령을 실행합니다.

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

이 새 프로파일에 노드를 할당합니다.

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

5.4. 노드의 ROOT 디스크 정의

일부 노드에서는 여러 디스크를 사용할 수 있습니다. 이러한 경우 **director**에서 프로비저닝 중에 **root** 디스크에 사용할 디스크를 식별해야 합니다. **director**가 **root** 디스크를 쉽게 식별할 수 있도록 다음과 같은 속성을 사용할 수 있습니다.

- **model**(문자열): 장치 식별자
- **vendor**(문자열): 장치 벤더

- **serial**(문자열): 디스크 일련 번호
- **hctl**(문자열): Host:Channel:Target:Lun (SCSI 용)
- **size**(정수): 장치의 크기(GB 단위)
- **wwn**(문자열): 고유한 스토리지 식별자
- **wwn_with_extension**(문자열): 벤더 확장이 첨부된 고유한 스토리지 식별자
- **wwn_vendor_extension**(문자열): 고유한 벤더 스토리지 식별자
- **rotational**(부울): 회전 장치인 경우(HDD) True, 그렇지 않은 경우 false(SSD)
- **name**(문자열): 장치의 이름(예: /dev/sdb1). 영구적인 이름이 있는 장치에만 사용

이 예에서는 **root** 장치를 식별하는 디스크의 일련 번호를 사용하여 오버클라우드 이미지를 배포할 드라이브를 지정합니다.

각 노드의 하드웨어 **introspection**에서 디스크 정보를 확인합니다. 다음 명령은 노드의 디스크 정보를 표시합니다.

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

예를 들면 하나의 노드의 데이터에서 세 개의 디스크가 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",
```

```

    "wnn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wnn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]

```

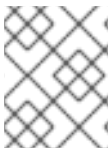
이 예에서는 **root** 장치를 일련 번호가 **61866da04f380d001ea4e13c12e36ad6**인 디스크 2로 설정합니다. 이를 위해 노드 정의에 **root_device** 매개 변수를 변경해야 합니다.

```

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

```

이를 통해 **director**가 **root** 디스크로 사용할 특정 디스크를 쉽게 식별할 수 있습니다. 오버클라우드 생성을 시작하면 **director**가 이 노드를 프로비저닝하고 오버클라우드 이미지를 이 디스크에 작성합니다.



참고

선택한 **root** 디스크에서 부팅을 포함하도록 각 노드의 **BIOS**를 설정합니다. 권장 부팅 순서는 네트워크 부팅, **root** 디스크 부팅 순입니다.



중요

name을 사용하여 **root** 디스크를 설정하지 마십시오. 노드가 부팅될 때 이 값이 변경될 수 있습니다.

5.5. 오버클라우드 사용자 정의

언더클라우드에는 오버클라우드 만들기 플랜 기능을 하는 **Heat** 템플릿 세트가 포함되어 있습니다. [고급 오버클라우드 사용자 정의](#) 가이드를 사용하여 오버클라우드의 고급 기능을 사용자 정의할 수 있습니다.

사용자 정의를 사용하지 않는 경우 기본 오버클라우드를 계속 배포할 수 있습니다. 자세한 내용은 [5.6절. “CLI 툴로 오버클라우드 생성”](#)에서 참조하십시오.



중요

기본 오버클라우드는 블록 스토리지에 로컬 **LVM** 스토리지를 사용하며, 이 설정은 지원되지 않습니다. 블록 스토리지에 외부 스토리지 솔루션을 사용하는 것이 좋습니다.

5.6. CLI 툴로 오버클라우드 생성

OpenStack 환경 생성의 최종 단계는 **openstack overcloud deploy** 명령을 실행하여 이 환경을 생성하는 것입니다. 이 명령을 실행하기 전에 주요 옵션 및 사용자 정의 환경 파일을 포함하는 방법을 숙지하고 있어야 합니다. 다음 섹션에서는 **openstack overcloud deploy** 명령 및 이 명령과 관련된 옵션에 대해 설명합니다.



주의

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

오버클라우드 매개 변수 설정

다음 표에는 **openstack overcloud deploy** 명령을 사용할 때의 추가 매개 변수가 나열되어 있습니다.

표 5.2. 배포 매개 변수

매개 변수	설명
--templates [TEMPLATES]	배포할 Heat 템플릿이 포함된 디렉터리. 비어 있을 경우 이 명령은 /usr/share/openstack-tripleo-heat-templates/ 를 기본 템플릿 위치로 사용합니다.
--stack STACK	생성하거나 업데이트할 스택의 이름
-t [TIMEOUT], --timeout [TIMEOUT]	배포 제한 시간 (분)
--libvirt-type [LIBVIRT_TYPE]	하이퍼바이저에 사용할 가상화 유형
--ntp-server [NTP_SERVER]	시간 동기화에 사용할 NTP(Network Time Protocol) 서버. 콤마로 구분된 목록에 여러 개의 NTP 서버를 지정할 수도 있습니다(예: --ntp-server 0.centos.pool.org,1.centos.pool.org). 고가용성 클러스터 배포를 위해서는 컨트롤러가 동일한 시간 소스를 일관되게 참조해야 합니다. 일반적인 환경에는 기존의 사례대로 이미 지정된 NTP 시간 소스가 있을 수 있습니다.
--no-proxy [NO_PROXY]	환경 변수 no_proxy 의 사용자 정의 값을 정의합니다. 이 변수는 프록시 통신에서 특정 호스트 이름을 제외합니다.
--overcloud-ssh-user OVERCLOUD_SSH_USER	오버클라우드 노드에 액세스할 SSH 사용자를 정의합니다. 일반적으로 SSH 액세스는 heat-admin 사용자를 통해 실행됩니다.

매개 변수	설명
-e [EXTRA HEAT TEMPLATE],--extra-template [EXTRA HEAT TEMPLATE]	오버클라우드 배포에 전달할 추가 환경 파일입니다. 두 번 이상 지정할 수 있습니다. openstack overcloud deploy 명령에 전달되는 환경 파일 순서가 중요합니다. 예를 들어 순차적으로 전달되는 각 환경 파일의 매개 변수는 이전 환경 파일의 동일한 매개 변수를 덮어씁니다.
--environment-directory	배포에 포함할 환경 파일이 들어 있는 디렉터리. 이 명령은 이러한 환경 파일을 먼저 숫자 순으로 처리한 다음 알파벳 순으로 처리합니다.
--validation-errors-nonfatal	오버클라우드 생성 프로세스에서는 일련의 사전 배포 검사를 수행합니다. 이 옵션은 사전 배포 검사에서 치명적이지 않은 오류가 발생할 경우에만 종료됩니다. 오류가 있으면 배포에 실패할 수 있으므로 이 옵션을 사용하는 것이 좋습니다.
--validation-warnings-fatal	오버클라우드 생성 프로세스는 일련의 사전 배포 검사를 수행합니다. 이 옵션은 사전 배포 검사에서 심각하지 않은 경고가 발생할 경우에만 종료됩니다.
--dry-run	오버클라우드에서 검증 확인을 수행하지만, 오버클라우드를 실제로 생성하지는 않습니다.
--skip-postconfig	오버클라우드 배포 후 설정을 건너뜁니다.
--force-postconfig	오버클라우드 배포 후 설정을 강제 적용합니다.
--answers-file ANSWERS_FILE	인수 및 매개 변수를 사용한 YAML 파일의 경로입니다.
--rhel-reg	오버클라우드 노드를 고객 포털 또는 Satellite 6 에 등록합니다.
--reg-method	오버클라우드 노드에 사용할 등록 방법입니다. Red Hat Satellite 6 또는 Red Hat Satellite 5 에는 satellite 를, 고객 포털에는 portal 을 사용합니다.
--reg-org [REG_ORG]	등록에 사용할 조직입니다.
--reg-force	시스템이 이미 등록되어 있어도 등록합니다.

매개 변수	설명
--reg-sat-url [REG_SAT_URL]	오버클라우드 노드를 등록할 Satellite 서버의 기본 URL입니다. Satellite의 HTTPS URL 대신 HTTP URL 을 이 매개 변수에 사용합니다. 예를 들어 https://satellite.example.com 대신 http://satellite.example.com 을 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 등록하고, katello-agent 를 설치합니다. Red Hat Satellite 5 서버인 경우 오버클라우드는 RHN-ORG-TRUSTED-SSL-CERT 파일을 가져오고, rhnreg_ks 에 등록합니다.
--reg-activation-key [REG_ACTIVATION_KEY]	등록에 사용할 활성화 키입니다.

일부 명령줄 매개 변수는 더 이상 사용되지 않으며, 대신 환경 파일의 **parameter_defaults** 섹션에 포함하는 Heat 템플릿 매개 변수가 사용됩니다. 다음 표에는 더 이상 사용되지 않는 매개 변수와 그에 상응하는 Heat 템플릿 매개 변수가 매핑되어 있습니다.

표 5.3. 더 이상 사용되지 않는 CLI 매개 변수와 상응하는 Heat 템플릿 매개 변수의 매핑

매개 변수	설명	Heat 템플릿 매개 변수
--control-scale	확장할 Controller 노드 수	ControllerCount
--compute-scale	확장할 Compute 노드 수	ComputeCount
--ceph-storage-scale	확장할 Ceph Storage 노드 수	CephStorageCount
--block-storage-scale	확장할 Cinder 노드 수	BlockStorageCount
--swift-storage-scale	확장할 Swift 노드 수	ObjectStorageCount
--control-flavor	Controller 노드에 사용할 플레이버	OvercloudControlFlavor
--compute-flavor	Compute 노드에 사용할 플레이버	OvercloudComputeFlavor
--ceph-storage-flavor	Ceph Storage 노드에 사용할 플레이버	OvercloudCephStorageFlavor
--block-storage-flavor	Cinder 노드에 사용할 플레이버	OvercloudBlockStorageFlavor

매개 변수	설명	Heat 템플릿 매개 변수
<code>--swift-storage-flavor</code>	Swift Storage 노드에 사용할 플레이트 이버	<code>OvercloudSwiftStorageFlavor</code>
<code>--neutron-flat-networks</code>	<code>neutron</code> 플러그인에 설정할 플랫폼 네트워크를 정의합니다. 외부 네트워크 생성을 허용하도록 기본적으로 "datacentre"로 설정됩니다.	<code>NeutronFlatNetworks</code>
<code>--neutron-physical-bridge</code>	각 하이퍼바이저에 생성할 Open vSwitch 브리지. 기본값은 "br-ex"입니다. 일반적으로 이 값은 변경할 필요가 없습니다.	<code>HypervisorNeutronPhysicalBridge</code>
<code>--neutron-bridge-mappings</code>	사용할 논리적 브리지와 물리적 브리지의 매핑. 기본적으로 호스트 (br-ex)의 외부 브리지가 물리적 이름(datacentre)에 매핑됩니다. 기본 유동 네트워크에 이 기본값을 사용합니다.	<code>NeutronBridgeMappings</code>
<code>--neutron-public-interface</code>	네트워크 노드의 br-ex에 브리지에 인터페이스를 정의합니다.	<code>NeutronPublicInterface</code>
<code>--neutron-network-type</code>	Neutron의 테넌트 네트워크 유형	<code>NeutronNetworkType</code>
<code>--neutron-tunnel-types</code>	Neutron 테넌트 네트워크의 터널 유형. 여러 값을 지정하려면 콤마로 구분된 문자열을 사용합니다.	<code>NeutronTunnelTypes</code>
<code>--neutron-tunnel-id-ranges</code>	테넌트 네트워크 할당에 사용할 수 있는 GRE 터널 ID 범위	<code>NeutronTunnelIdRanges</code>
<code>--neutron-vni-ranges</code>	테넌트 네트워크 할당에 사용할 수 있는 VXLAN VNI ID 범위	<code>NeutronVniRanges</code>
<code>--neutron-network-vlan-ranges</code>	지원할 Neutron ML2 및 Open vSwitch VLAN 매핑 범위. 기본적으로 datacentre 물리적 네트워크에서 VLAN을 허용하도록 설정되어 있습니다.	<code>NeutronNetworkVLANRanges</code>
<code>--neutron-mechanism-drivers</code>	<code>neutron</code> 테넌트 네트워크의 매커니즘 드라이버. 기본값은 "openvswitch"입니다. 여러 값을 지정하려면 콤마로 구분된 문자열을 사용합니다.	<code>NeutronMechanismDrivers</code>

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

이러한 매개 변수는 Red Hat OpenStack Platform의 이후 버전에서 삭제될 예정입니다.



참고

전체 옵션 목록을 표시하려면 다음 명령을 실행하십시오.

```
$ openstack help overcloud deploy
```

5.7. 오버클라우드 생성에 환경 파일 추가

-e에 오버클라우드를 사용자 정의할 환경 파일이 포함되어 있습니다. 필요에 따라 환경 파일을 추가할 수 있지만, 나중에 실행되는 환경 파일에 정의된 매개 변수와 리소스가 우선 순위를 갖기 때문에 환경 파일의 순서가 중요합니다. 다음 목록을 환경 파일 순서의 예로 사용하십시오.

- 각 역할 및 해당 플레이어당 노드의 크기입니다. 오버클라우드 생성에 이 정보를 포함하는 것이 중요합니다.
- **heat** 템플릿 컬렉션의 초기화 파일(**environments/network-isolation.yaml**)부터 시작하여 네트워크 분리 파일, 사용자 정의 **NIC** 설정 파일, 마지막으로 추가 네트워크 설정 순입니다.
- 모든 외부 로드 밸런싱 환경 파일
- **Ceph Storage**, **NFS**, **iSCSI** 등과 같은 스토리지 환경 파일
- **Red Hat CDN** 또는 **Satellite** 등록의 환경 파일
- 기타 사용자 정의 환경 파일

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

```
$ 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 \
```

이 명령에는 다음 추가 옵션이 포함되어 있습니다.

- **--templates - /usr/share/openstack-tripleo-heat-templates**의 Heat 템플릿 컬렉션을 사용하여 오버클라우드를 생성합니다.
- **-e ~/templates/node-info.yaml - -e** 옵션은 추가 환경 파일을 오버클라우드 배포에 추가합니다. 이 경우에는 각 역할에 사용할 노드 수와 플레이버를 정의하는 사용자 정의 환경 파일입니다. 예를 들면 다음과 같습니다.

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

- **-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml - -e** 옵션은 추가 환경 파일을 오버클라우드 배포에 추가합니다. 이 경우에는 네트워크 분리 설정을 초기화하는 환경 파일입니다.
- **-e ~/templates/network-environment.yaml - -e** 옵션은 추가 환경 파일을 오버클라우드 배포에 추가합니다. 이 경우에는 네트워크 분리에 대한 사용자 정의가 들어 있는 네트워크 환경 파일입니다.
- **-e ~/templates/storage-environment.yaml - -e** 옵션은 추가 환경 파일을 오버클라우드 배포에 추가합니다. 이 경우에는 스토리지 구성을 초기화하는 사용자 정의 환경 파일입니다.
- **--ntp-server pool.ntp.org** - 시간 동기화에 NTP 서버를 사용합니다. Controller 노드 클러스터를 동기화 상태로 유지하려는 경우 유용합니다.

director를 사용하려면 [8장. 오버클라우드 생성 후 작업 수행](#)에 재배포 및 배포 후 기능에 대한 이러한 환경 파일이 필요합니다. 이러한 파일이 포함되지 않은 경우 오버클라우드가 손상될 수 있습니다.

오버클라우드 설정을 나중에 수정하려는 경우 다음을 수행해야 합니다.

1. 사용자 정의 환경 파일 및 Heat 템플릿에서 매개 변수 수정
2. 같은 환경 파일을 지정하고 **openstack overcloud deploy** 명령을 다시 실행

오버클라우드 설정을 직접 편집하지 마십시오. 오버클라우드 스택을 director로 업데이트할 때 수동 설정을 director의 설정이 덮어쓰기하므로 설정을 직접 편집하지 마십시오.

환경 파일 디렉터리 추가

--environment-directory 옵션을 사용하여 환경 파일이 포함된 전체 디렉터리 전체를 추가할 수 있습니다. 배포 명령은 이 디렉터리에 있는 환경 파일을 먼저 숫자 순으로 처리한 다음 알파벳 순으로 처리합니다. 이 방법을 사용하는 경우 파일 이름과 함께 숫자 접두어를 사용하여 처리 순서를 지정하는 것이 좋습니다. 예를 들면 다음과 같습니다.

```
$ ls -1 ~/templates
00-node-info.yaml
10-network-isolation.yaml
```

```
20-network-environment.yaml
30-storage-environment.yaml
40-rhel-registration.yaml
```

다음 배포 명령을 실행하여 디렉터리를 추가합니다.

```
$ openstack overcloud deploy --templates --environment-directory
~/templates
```

응답 파일 사용

응답 파일은 템플릿과 환경 파일을 간편하게 추가할 수 있도록 하는 YAML 형식의 파일입니다. 응답 파일은 다음 매개 변수를 사용합니다.

templates

사용할 코어 Heat 템플릿 컬렉션으로, **--templates** 명령줄 옵션을 대체하는 역할을 합니다.

environments

추가할 환경 파일 목록으로, **--environment-file (-e)** 명령줄 옵션을 대체하는 역할을 합니다.

예를 들면 응답 파일에 다음이 포함될 수 있습니다.

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

다음 배포 명령을 실행하여 응답 파일을 추가합니다.

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

5.8. 오버클라우드 플랜 관리

openstack overcloud deploy 명령을 사용하지 않고, **director**에서 가져온 플랜을 관리할 수도 있습니다.

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

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

이 명령을 실행하면 코어 Heat 템플릿 컬렉션의 플랜이 **/usr/share/openstack-tripleo-heat-templates**에 생성됩니다. **director**는 입력된 내용을 기반으로 하여 플랜에 이름을 지정합니다. 이 예의 경우 **my-overcloud**입니다. **director**는 이 이름을 오브젝트 스토리지 컨테이너, 워크플로우 환경 및 오버클라우드 스택 이름에 대한 레이블로 사용합니다.

다음 명령을 사용하여 환경 파일의 매개 변수를 추가합니다.

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

다음 명령을 사용하여 플랜을 배포합니다.

```
$ openstack overcloud plan deploy my-overcloud
```

다음 명령을 사용하여 기존 플랜을 삭제합니다.

```
$ openstack overcloud plan delete my-overcloud
```



참고

openstack overcloud deploy 명령은 기본적으로 이러한 명령을 모두 사용하여 기존 플랜을 제거하고, 환경 파일과 함께 새 플랜을 업로드하고, 플랜을 배포합니다.

5.9. 오버클라우드 템플릿 및 플랜 검증

오버클라우드 생성 또는 스택 업데이트를 실행하기 전에 **Heat** 템플릿 및 환경 파일에서 오류를 확인합니다.

렌더링된 템플릿 생성

오버클라우드의 코어 **Heat** 템플릿은 **Jinja2** 포맷으로 되어 있습니다. 템플릿을 확인하려면 다음 명령을 사용하여 **Jinja2** 포맷을 사용하지 않고 버전을 렌더링합니다.

```
$ openstack overcloud plan create --templates /usr/share/openstack-tripleo-heat-templates overcloud-validation
$ mkdir ~/overcloud-validation
$ cd ~/overcloud-validation
$ swift download overcloud-validation
```

다음에 나오는 검증 테스트에 **~/overcloud-validation**에 있는 렌더링된 템플릿을 사용하십시오.

템플릿 구문 확인

다음 명령을 사용하여 템플릿 구문을 확인합니다.

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



참고

검증을 수행하려면 오버클라우드 특정 리소스를 포함할 **overcloud-resource-registry-puppet.yaml** 환경 파일이 필요합니다. **-e** 옵션을 사용하여 이 명령에 환경 파일을 추가하십시오. 또한 **--show-nested** 옵션을 추가하여 중첩된 템플릿에서 매개 변수를 해결합니다.

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

5.10. 오버클라우드 생성 모니터링

오버클라우드 생성 절차가 시작되고 **director**에서 노드를 프로비저닝합니다. 이 프로세스는 완료하는 데 시간이 걸립니다. 오버클라우드 생성 상태를 보려면 별도의 터미널을 **stack** 사용자로 열고 다음을 실행합니다.

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

openstack stack list --nested 명령을 실행하면 오버클라우드 생성의 현재 단계가 표시됩니다.

5.11. 오버클라우드 액세스

director에서는 **director** 호스트에서 오버클라우드와 상호 작용을 설정하고 인증을 지원하는 스크립트를 생성합니다. **director**는 **overcloudrc** 파일을 **stack** 사용자의 홈 **director**에 저장합니다. 이 파일을 사용하려면 다음 명령을 실행합니다.

```
$ source ~/overcloudrc
```

이 명령을 실행하면 **director** 호스트의 CLI에서 오버클라우드와 상호 작용하는 데 필요한 환경 변수가 로드됩니다. **director** 호스트와의 상호 작용으로 돌아가려면 다음 명령을 실행합니다.

```
$ source ~/stackrc
```

오버클라우드의 각 노드에는 **heat-admin**이라고 하는 사용자가 포함되어 있습니다. **stack** 사용자는 각 노드에 있는 이 사용자에게 대해 SSH 액세스 권한을 갖습니다. SSH로 노드에 액세스하려면 원하는 노드의 IP 주소를 확인합니다.

```
$ nova list
```

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

```
$ ssh heat-admin@192.168.24.23
```

5.12. 오버클라우드 생성 완료

이제 명령줄 툴을 사용한 오버클라우드 생성이 완료되었습니다. 생성 이후 기능에 대해서는 [8장. 오버클라우드 생성 후 작업 수행](#)을 참조하십시오.

6장. 웹 UI로 기본 오버클라우드 설정

이 장에서는 웹 UI를 사용하여 OpenStack Platform 환경의 기본 설정 단계를 설명합니다. 기본 설정을 사용하는 오버클라우드에는 사용자 정의 기능이 없지만 고급 설정 옵션을 이 기본 오버클라우드에 추가하고, [고급 오버클라우드 사용자 정의](#) 가이드에 나와 있는 지침에 따라 사양에 맞게 사용자 정의를 할 수 있습니다.

이 장의 예에서는 모든 노드가 전원 관리에 IPMI를 사용하는 베어 메탈 시스템으로 되어 있습니다. 지원되는 전원 관리 유형 및 옵션에 대한 자세한 내용은 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

워크플로우

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

요구 사항

- [4장. 언더클라우드 설치](#)에서 생성한 UI가 활성화되어 있는 director 노드
- 노드에 사용하는 베어 메탈 머신 세트. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다(오버클라우드 역할에 대한 자세한 내용은 [3.1절. “노드 배포 역할 플래닝”](#) 참조). 이러한 머신은 각 노드 유형에 대해 설정된 요구 사항을 준수해야 합니다. 이러한 요구 사항은 [2.4절. “오버클라우드 요구 사항”](#)을 참조하십시오. 이러한 노드에는 운영 체제가 필요하지 않습니다. director에서 Red Hat Enterprise Linux 7 이미지를 각 노드에 복사합니다.
- 기본 VLAN으로 구성된 프로비저닝 네트워크에 대한 한 개의 네트워크 연결. 모든 노드는 이 네트워크에 연결해야 하며, [2.3절. “네트워킹 요구 사항”](#)에 설정된 요구 사항에 부합해야 합니다.
- 다른 모든 네트워크 유형은 OpenStack 서비스에 프로비저닝 네트워크를 사용하지만 다른 네트워크 트래픽 유형에 대해 추가로 네트워크를 생성할 수 있습니다.

6.1. 웹 UI에 액세스

사용자는 SSL을 통해 director의 웹 UI에 액세스합니다. 예를 들어 언더클라우드의 IP 주소가 192.168.24.1 이면 UI에 액세스할 주소는 <https://192.168.24.1>입니다. 웹 UI는 처음에 다음 필드가 포함된 로그인 화면을 표시합니다.

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

UI에 로그인하면 UI에서 OpenStack Identity Public API에 액세스하고 다른 공용 API 서비스의 엔드포인트를 가져옵니다. 이러한 서비스에는 다음이 포함됩니다.

구성 요소	UI 용도
OpenStack Identity(keystone)	UI에 대한 인증 및 다른 서비스의 엔드포인트 검색에 사용됩니다.

구성 요소	UI 용도
OpenStack Orchestration(heat)	배포 상태를 위해 사용됩니다.
OpenStack Bare Metal(ironic)	노드 제어에 사용됩니다.
OpenStack Object Storage(swift)	Heat 템플릿 컬렉션 또는 오버클라우드 생성에 사용되는 플랜 스토리지를 위해 사용됩니다.
OpenStack Workflow(mistral)	director 작업에 액세스하고 실행합니다.
OpenStack Messaging(zaqar)	특정 작업의 상태를 검색하기 위한 websocket 기반 서비스입니다.

클라이언트 시스템은 해당 엔드포인트에 액세스해야 하기 때문에 UI는 이러한 공용 API와 직접 상호 작용합니다. **director**는 SSL/TLS 암호화 경로를 통해 이러한 엔드포인트를 공용 VIP(**undercloud.conf** 파일의 **undercloud_public_host**)에 노출합니다. 각 경로는 서비스에 대응합니다. 예를 들면 **https://192.168.24.2:443/keystone**은 OpenStack Identity 공용 API에 매핑됩니다.

엔드포인트를 변경하거나 엔드포인트 액세스에 다른 IP를 사용하려는 경우 **director** UI가 **/var/www/openstack-tripleo-ui/dist/tripleo_ui_config.js** 파일에서 설정을 읽습니다. 이 파일은 다음 매개 변수를 사용합니다.

매개 변수	설명
keystone	OpenStack Identity(keystone) 서비스에 대한 공용 API. UI는 이 서비스를 통해 다른 서비스에 대한 엔드포인트를 자동으로 검색하므로 이 매개 변수를 정의하기만 하면 됩니다. 하지만 필요한 경우 다른 엔드포인트의 사용자 정의 URL을 정의할 수 있습니다.
heat	OpenStack Orchestration(heat) 서비스에 대한 공용 API.
ironic	OpenStack Bare Metal(ironic) 서비스에 대한 공용 API.
swift	OpenStack Object Storage(swift) 서비스에 대한 공용 API.
mistral	OpenStack Workflow(mistral) 서비스에 대한 공용 API.
zaqar-websocket	OpenStack Messaging(zaqar) 서비스에 대한 websocket .
zaqar_default_queue	OpenStack Messaging(zaqar) 서비스에 사용할 메시지 큐. 기본값은 tripleo 입니다.

다음은 **192.168.24.2**가 언더클라우드의 공용 VIP인 **tripleo-ui-config.js** 파일의 예입니다.

```
window.tripleoUiConfig = {
  'keystone': 'https://192.168.24.2:443/keystone/v2.0',
  'heat': 'https://192.168.24.2:443/heat/v1/(tenant_id)s',
  'ironic': 'https://192.168.24.2:443/ironic',
  'mistral': 'https://192.168.24.2:443/mistral/v2',
  'swift': 'https://192.168.24.2:443/swift/v1/AUTH_(tenant_id)s',
  'zaqar-websocket': 'wss://192.168.24.2:443/zaqar',
  'zaqar_default_queue': "tripleo"
};
```

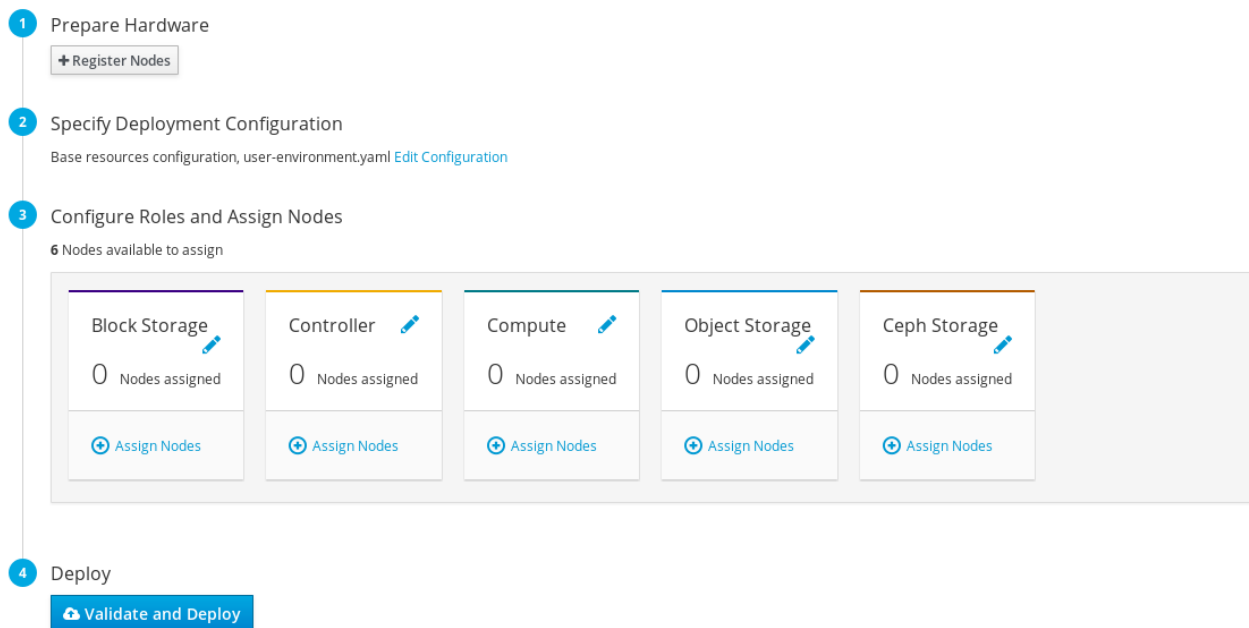
6.2. 웹 UI 탐색

UI는 다음 세 가지 섹션으로 구성되어 있습니다.

배포 플랜

UI 상단에 있는 메뉴 항목. 이 페이지는 주요 UI 섹션으로 기능하며, 오버클라우드 생성에 사용할 플랜, 각 역할에 할당할 노드 및 현재 오버클라우드의 상태를 정의할 수 있습니다. 이 섹션에서는 배포 매개 변수 설정 및 역할에 대한 노드 할당을 비롯하여 오버클라우드 생성 프로세스를 단계별로 설명합니다.

overcloud [Manage Deployments](#)



노드

UI 상단에 있는 메뉴 항목. 이 페이지는 노드 설정 섹션 역할을 하며, 새 노드를 등록하고 등록된 노드를 introspect할 방법을 제공합니다. 또한 배포에 사용할 수 있는 노드, 현재 배포된 노드 및 유지보수 중인 노드를 정의합니다.

Nodes

Refresh Results + Register Nodes

Registered 8		Deployed 0	Maintenance 0						
Filter		Showing 8 of 8 items						Introspect Nodes	Provide Nodes
	MAC Address(es)	Name	Role	CPU Arch.	CPU (cores)	Disk (GB)	Memory (MB)	Power State	Provision State
<input type="checkbox"/>	52:54:00:ec:52:50	node01	Not assigned	x86_64	1	49	4096	power off	available
<input type="checkbox"/>	52:54:00:62:22:24	node02	Not assigned	x86_64	1	49	4096	power off	available
<input type="checkbox"/>	52:54:00:d7:f9:fa	node03	Not assigned	x86_64	2	99	8192	power off	available
<input type="checkbox"/>	52:54:00:1a:8c:ee	node04	Not assigned	x86_64	2	99	8192	power off	available
<input type="checkbox"/>	52:54:00:09:b0:ff	node05	Not assigned	x86_64	2	99	8192	power off	available
<input type="checkbox"/>	52:54:00:5a:6c:b9	node06	Not assigned	x86_64	1	49	6144	power off	available
<input type="checkbox"/>	52:54:00:d8:d4:1e	node07	Not assigned	x86_64	1	49	8192	power off	available
<input type="checkbox"/>	52:54:00:b8:fe:fb	node08	Not assigned	x86_64	1	49	4096	power off	available

검증

페이지 오른쪽에 있는 측면 패널. 이 섹션에서는 오버클라우드 생성 프로세스가 성공적으로 실행되도록 배포 전 및 배포 후 시스템 검사를 수행합니다. 이러한 검증 작업은 배포 시 특정 시점에서 자동으로 실행되지만 수동으로 실행할 수도 있습니다. 실행하려는 검증 작업의 **재생** 버튼을 클릭합니다. 각 검증 작업의 제목을 클릭하여 실행하거나, 검증 제목을 클릭하여 자세한 정보를 표시합니다.



Verify the undercloud fits the RAM requirements

Verify that the undercloud has enough RAM. <https://access.redhat.com/...>

prep

pre-introspection

6.3. 웹 UI에서 오버클라우드 플랜 가져오기

director UI에는 오버클라우드 구성 전 플랜이 필요합니다. 이 플랜은 일반적으로 `/usr/share/openstack-tripleo-heat-templates`의 언더클라우드에 있는 것과 같은 Heat 템플릿 컬렉션입니다. 또한 하드웨어 및 환경 요구 사항에 맞게 플랜을 사용자 정의할 수 있습니다. 오버클라우드 사용자 정의에 대한 자세한 내용은 [고급 오버클라우드 사용자 정의](#) 가이드를 참조하십시오.

플랜에는 오버클라우드 설정에 대한 네 가지 주요 단계가 표시됩니다.

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

언더클라우드 설치 및 설정을 실행하면 플랜이 자동으로 업로드됩니다. 여러 플랜을 웹 UI에 가져올 수도 있습니다. 배포 플랜 화면에서 **배포 관리**를 클릭하면 현재 플랜 테이블이 표시됩니다.

새로운 플랜 생성을 클릭하면 다음 정보를 묻는 창이 표시됩니다.

- **플랜 이름** - 플랜의 일반 텍스트 이름입니다(예: **overcloud**).
- **업로드 유형** - **Tar 압축(tar.gz)**을 업로드할 것인지 아니면 **로컬 폴더 전체(Google Chrome 전용)**를 업로드할 것인지 선택합니다.

- **플랜 파일** - 브라우저를 클릭하여 로컬 파일 시스템에서 플랜을 선택합니다.

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

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

director UI에 플랜이 업로드되면 플랜이 **Plans** 테이블에 표시되어 설정할 수 있습니다. **배포 플랜**을 클릭합니다.

Plans

Showing 1 of 1 items		+ Create New Plan
Name	Actions	
overcloud	Edit	Delete

6.4. 웹 UI에서 노드 등록

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

- **배포 플랜** 화면에서 **1 하드웨어 준비**에 있는 **노드 등록**을 클릭합니다.
- **노드** 화면에서 **노드 등록**을 클릭합니다.

이렇게 하면 노드 등록 창이 표시됩니다.

director에는 등록할 노드 목록이 필요하며 다음 두 가지 방법 중 하나를 사용하여 제공할 수 있습니다.

1. 노드 정의 템플릿 업로드 - 이를 위해서는 **파일 업로드** 버튼을 클릭하고 파일을 선택해야 합니다. 노드 정의 템플릿 구문에 대해서는 [5.1절. “오버클라우드에 노드 등록”](#)을 참조하십시오.
2. 수동으로 각 노드 등록 - 이를 위해서는 **새로 추가**를 클릭하고 노드에 대한 세부 사항을 제공해야 합니다.

수동 등록을 위해 제공해야 하는 세부 사항은 다음과 같습니다.

이름

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

드라이버

사용할 전원 관리 드라이버. 이 예에서는 IPMI 드라이버(**pxe_ipmitool**)를 사용하지만, 다른 드라이버를 사용할 수도 있습니다. 사용 가능한 드라이버에 대해서는 [부록 B. 전원 관리 드라이버](#)를 참조하십시오.

IPMI IP 주소

IPMI 장치의 IP 주소입니다.

IPMI 사용자 이름, IPMI 암호

IPMI 사용자 이름 및 암호입니다.

아키텍처

(선택 사항) 시스템 아키텍처입니다.

CPU 수

(선택 사항) 노드에 있는 CPU 수입니다.

메모리(MB)

(선택 사항) 메모리 크기(MB)입니다.

디스크(GB)

(선택 사항) 하드 디스크의 크기(GB)입니다.

NIC MAC 주소

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

**참고**

UI를 통해 **pxe_ssh** 드라이버를 사용하여 KVM 호스트에서 노드를 등록할 수 있습니다. 이 옵션은 테스트 및 평가 목적으로만 사용할 수 있습니다. Red Hat OpenStack Platform 엔터프라이즈 환경에는 권장되지 않습니다. 자세한 내용은 [B.7절. “SSH 및 Virsh”](#)를 참조하십시오.

노드 정보를 입력한 후 창 하단에 있는 **노드 등록**을 클릭합니다.

director가 노드를 등록합니다. 완료되면 UI를 사용하여 노드에서 **introspection**을 수행할 수 있습니다.

6.5. 웹 UI에서 노드의 하드웨어 검사

director UI에서 각 노드의 **introspection** 프로세스를 실행할 수도 있습니다. 이 프로세스를 실행하면 각 노드가 PXE로 **introspection** 에이전트를 부팅합니다. 이 에이전트는 노드에서 하드웨어 데이터를 수집하고 director에 다시 보냅니다. 그러면 director가 이 **introspection** 데이터를 director에서 실행 중인 OpenStack Object Storage(swift) 서비스에 저장합니다. director는 프로필 태그, 벤치마킹, 수동 root 디스크 할당과 같은 다양한 목적으로 하드웨어 정보를 사용합니다.

**참고**

또한 정책 파일을 생성하여 **introspection** 직후 노드를 프로필에 자동으로 태그할 수도 있습니다. 정책 파일 생성 및 **introspection** 프로세스에 해당 파일을 통합하는 작업에 대한 자세한 내용은 [부록 E. 자동 프로필 태깅](#)을 참조하십시오. 또는 UI를 통해 프로필에 노드를 태그할 수 있습니다. 노드를 수동으로 태그하는 작업에 대한 자세한 내용은 [6.7절. “웹 UI에서 역할에 노드 태그”](#)을 참조하십시오.

introspection 프로세스를 시작하려면 다음 단계를 수행합니다:

1. 노드 화면으로 이동합니다.
2. **introspect**할 노드를 모두 선택합니다.
3. **Introspect** 노드를 클릭합니다.

**중요**

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

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

이제 노드를 프로비저닝할 준비가 되었습니다.

6.6. 웹 UI에서 오버클라우드 플랜 매개 변수 편집

배포 플랜 화면에서는 업로드된 플랜을 사용자 정의할 수 있습니다. **2 배포 설정 지정**에서 **설정 편집** 링크를 클릭하여 기본 오버클라우드 설정을 변경합니다.

다음 두 개의 기본 탭이 있는 창이 표시됩니다.

전체 설정

이 탭에서는 오버클라우드의 다른 기능을 추가할 수 있습니다. 이러한 기능은 플랜의 **capabilities-map.yaml** 파일에 정의되어 있으며 이러한 기능은 각각 다른 환경 파일을 사용합니다. 예를 들어, 스토리지에서 플랜이 **environments/storage-environment.yaml** 파일에 매핑되고 오버클라우드의 NFS, iSCSI 또는 Ceph 설정을 구성할 수 있는 **스토리지 환경**을 선택할 수 있습니다. 기타 탭에는 플랜에서 감지되었지만 **capabilities-map.yaml(plan)**에 포함된 사용자 정의 환경 파일을 추가하는 데 유용함)에 나열되지 않은 환경 파일이 들어 있습니다. 추가할 기능을 선택한 후 **저장**을 클릭합니다.

Deployment Configuration

Overall Settings

Parameters

Deployment Options

Nova Extensions

Utilities

Base Resources Configuration

Operational Tools

Neutron Plugin Configuration

Other

Overlay Network Configuration

Additional Services

Storage

High Availability

Enables configuration of an Overcloud controller with Pacemaker

☐ Pacemaker

Enable configuration of an Overcloud controller with Pacemaker

Pacemaker options

☐ Pacemaker No Restart

Docker RDO

Docker container with heat agents for containerized compute node

☐ Docker RDO

Enable TLS

☐ TLS

Use this option to pass in certificates for SSL deployments. For these values to take effect, one of the TLS endpoints environments must also be used.

TLS Endpoints

☐ SSL-enabled deployment with DNS name as public endpoint

Use this environment when deploying an SSL-enabled overcloud where the public endpoint is a DNS name.

☐ SSL-enabled deployment with IP address as public endpoint

Use this environment when deploying an SSL-enabled overcloud where the public endpoint is an IP address.

External load balancer

Enable external load balancer

☐ External load balancer IPv6
 ☐ External load balancer IPv4

Save Changes

Cancel

매개 변수

여기에는 오버클라우드에 대한 여러 가지 기본 수준 및 환경 파일 매개 변수가 포함되어 있습니다. 예를 들면 이 섹션의 각 역할에 대한 노드 수를 변경할 수 있습니다. 세 개의 **Controller** 노드를 사용하려는 경우 **ControllerCount**를 3으로 변경합니다. 기본 수준 매개 변수를 수정했으면 **저장**을 클릭합니

다.

Deployment Configuration
X

Overall Settings
Parameters

BlockStorageCount
0
Number of BlockStorage nodes to deploy

BlockStorageHostnameFormat
%stackname%-blockstorage-%index%
Format for BlockStorage node hostnames Note %index% is translated into the index of the node, e.g 0/1/2 etc and %stackname% is replaced with the stack name e.g overcloud

BlockStorageRemovalPolicies
[]
List of resources to be removed from BlockStorage ResourceGroup when doing an update which requires removal of specific resources. Example format ComputeRemovalPolicies: [{resource_list: ['0']}]

BlockStorageSchedulerHints
{}
Optional scheduler hints to pass to nova

BlockStorageServices
OS::TripleO::Services::CACerts,OS::TripleO::Services::BlockStorageCinderVolume,OS::TripleO::Services::Kernel,OS::TripleO::Services::Ntp,OS::TripleO::Services::Timezone,OS::TripleO::Services::Snmp,OS::TripleO::Services::TripleoPackages,OS::TripleO::Services::TripleoFirewall,OS::TripleO::Services::SensuClient,OS::TripleO::Services::FluentdClient,OS::TripleO::Services::VipHosts
A list of service resources (configured in the Heat resource_registry) which represent nested stacks for each service that should get installed on the BlockStorage role.

CephStorageCount
0
Number of CephStorage nodes to deploy

CephStorageHostnameFormat
%stackname%-cephstorage-%index%
Format for CephStorage node hostnames Note %index% is translated into the index of the node, e.g 0/1/2 etc and %stackname% is replaced with the stack name e.g overcloud

6.7. 웹 UI에서 역할에 노드 태그

각 노드의 하드웨어 등록 및 검사 후 특정 프로필에 태그합니다. 이러한 프로필은 노드를 특정 유형 및 배포 역할과 일치시킵니다.

노드를 역할에 할당하려면 **배포 플랜** 화면에서 **3 역할 구성 및 노드 할당** 섹션으로 스크롤합니다. 선택한 역할에 대해 **노드 할당**을 클릭합니다. 역할에 할당할 노드를 선택할 수 있는 창이 표시됩니다. 역할 노드를 선택했으면 **선택 노드를 할당/할당 해제**를 클릭합니다.

Assign Nodes to Controller Role ✕

Showing 8 of 8 items
Assign/Unassign Selected Nodes

	MAC Address(es)	Name	Role	CPU Arch.	CPU (cores)	Disk (GB)	Memory (MB)	Power State	Provision State
<input type="checkbox"/>	52:54:00:b8:fe:fb	node08	Not assigned	x86_64	1	49	4096	power off	available
<input type="checkbox"/>	52:54:00:1a:8c:ee	node04	Not assigned	x86_64	2	99	8192	power off	available
<input type="checkbox"/>	52:54:00:5a:6c:b9	node06	Not assigned	x86_64	1	49	6144	power off	available
<input type="checkbox"/>	52:54:00:d7:f9:fa	node03	Not assigned	x86_64	2	99	8192	power off	available
<input type="checkbox"/>	52:54:00:ec:52:50	node01	Not assigned	x86_64	1	49	4096	power off	available
<input type="checkbox"/>	52:54:00:09:b0:ff	node05	Not assigned	x86_64	2	99	8192	power off	available
<input type="checkbox"/>	52:54:00:d8:d4:1e	node07	Not assigned	x86_64	1	49	8192	power off	available
<input type="checkbox"/>	52:54:00:62:22:24	node02	Not assigned	x86_64	1	49	4096	power off	available

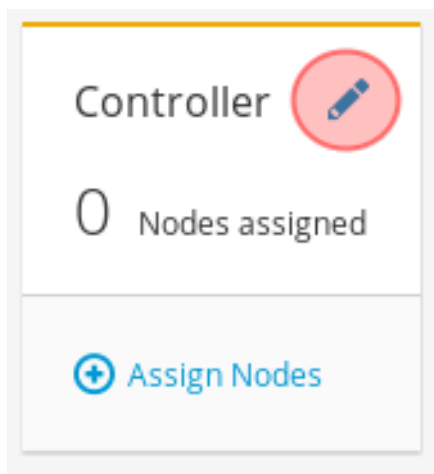
Done

이러한 노드가 역할에 할당되면 **완료**를 클릭하여 **배포 플랜** 화면으로 돌아갑니다.

오버클라우드에 포함시키려는 각 역할에 대해 이 작업을 완료합니다.

6.8. 웹 UI에서 노드 편집

각 노드 역할은 역할별 매개 변수를 구성하는 방법을 제공합니다. **배포 플랜** 화면에서 **3 역할 구성 및 노드 할당** 역할로 스크롤합니다. 역할 이름 옆에 있는 **역할 매개 변수 편집** 아이콘(연필 아이콘)을 클릭합니다.



다음 두 가지 기본 탭이 표시된 창이 나타납니다.

매개 변수

여기에는 여러 역할별 매개 변수가 포함되어 있습니다. 예를 들어 컨트롤러 역할을 편집하는 경우 **OvercloudControlFlavor** 매개 변수를 사용하여 역할에 대한 기본 유형을 변경할 수 있습니다. 역할별 매개 변수를 수정한 후 **변경사항 저장**을 클릭합니다.

Controller Role
X

Parameters
Services
Network Configuration

SoftwareConfigTransport

POLL_TEMP_URL

How the server should receive the metadata required for software configuration.

EndpointMap

{}

Mapping of service endpoint -> protocol. Typically set via parameter_defaults in the resource registry.

HostnameMap

{}

Optional mapping to override hostnames

ConfigCommand

os-refresh-config --timeout 14400

Command which will be run whenever configuration data changes

KeyName

default

Name of an existing Nova key pair to enable SSH access to the instances

NetworkDeploymentActions

CREATE

Heat action when to apply network configuration changes

ControllerSchedulerHints

{}

Optional scheduler hints to pass to nova

서비스

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

Controller Role

Parameters

Services

Network Configuration

OS::TripleO::Services::NeutronCorePlugin
OS::TripleO::Services::SwiftRingBuilder
OS::TripleO::Services::CeilometerCollector
OS::TripleO::Services::NovaApi
OS::TripleO::Services::GnocchiStatsd
OS::TripleO::Services::AodhNotifier
OS::TripleO::Services::Timezone
OS::TripleO::Services::NovaScheduler
OS::TripleO::Services::NeutronDhcpAgent
OS::TripleO::Services::CinderApi
OS::TripleO::Services::NeutronMetadataAgent
OS::TripleO::Services::SwiftProxy
OS::TripleO::Services::TripleoFirewall
OS::TripleO::Services::NovaMetadata
OS::TripleO::Services::CeilometerAgentNotification

TimeZone

UTC

The timezone to be set on the overcloud.

DefaultPasswords

{}

EndpointMap

{}

Mapping of service endpoint -> protocol. Typically set via parameter_defaults in the resource registry.

ServiceNetMap

{}

Mapping of service_name -> network name. Typically set via parameter_defaults in the resource registry. This mapping overrides those in ServiceNetMapDefaults.

네트워크 설정

네트워크 설정을 통해 오버클라우드의 여러 네트워크에 대한 IP 주소 또는 서브넷 범위를 정의할 수 있습니다.

Controller Role

Parameters

Services

Network Configuration

Software Config to drive os-net-config for a simple bridge.

TenantIpSubnet

IP address/subnet on the tenant network

ManagementIpSubnet

IP address/subnet on the management network

ExternalIpSubnet

IP address/subnet on the external network

StorageIpSubnet

IP address/subnet on the storage network

StorageMgmtIpSubnet

IP address/subnet on the storage mgmt network

ControlPlaneIp

IP address/subnet on the ctlplane network

InternalApiIpSubnet

IP address/subnet on the internal API network



중요

역할 서비스 매개 변수가 UI에 표시되더라도 일부 서비스가 기본적으로 표시되지 않을 수 있습니다. 이러한 서비스는 [6.6절. “웹 UI에서 오버클라우드 플랜 매개 변수 편집”](#)의 지침을 통해 활성화할 수 있습니다. 이러한 서비스 활성화에 대한 자세한 내용은 [고급 오버클라우드 사용자 정의 가이드의 작성 가능한 역할](#) 섹션을 참조하십시오.

6.9. 웹 UI에서 오버클라우드 생성 시작

오버클라우드 플랜이 구성되면 오버클라우드 배포를 시작할 수 있습니다. 이를 위해서는 **4 배포** 섹션으로 스크롤하여 **검증 및 배포**를 클릭합니다.

 Validate and Deploy

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



Deploy Plan overcloud

Summary: Base resources configuration, user-environment.yaml



Not all pre-deployment validations have passed

It is highly recommended that you resolve all validation issues before continuing.

Are you sure you want to deploy this plan?

 Deploy

배포할 준비가 되었으면 **배포**를 클릭합니다.

UI에서는 오버클라우드의 생성 진행률을 정기적으로 모니터링하고 현재 진행률을 나타내는 진행률 표시줄을 표시합니다. [자세한 정보 보기](#) 링크는 오버클라우드의 현재 **OpenStack Orchestration** 스택 로그를 표시합니다.

Plan overcloud deployment
✕

Deployment in progress
31%

Resources

Filter

Showing 52 of 52 items

Name	Status	Updated Time
MysqlRootPassword	CREATE_COMPLETE	2016-11-24T07:00:08Z
PcsdPassword	CREATE_COMPLETE	2016-11-24T07:00:08Z
VipMap	CREATE_COMPLETE	2016-11-24T07:00:08Z
RabbitCookie	CREATE_COMPLETE	2016-11-24T07:00:08Z
Controller	INIT_COMPLETE	2016-11-24T07:00:08Z
ObjectStorage	INIT_COMPLETE	2016-11-24T07:00:08Z
ObjectStorageIplListMap	INIT_COMPLETE	2016-11-24T07:00:08Z
ControllerIplListMap	INIT_COMPLETE	2016-11-24T07:00:08Z
BlockStorageServiceChain	CREATE_IN_PROGRESS	2016-11-24T07:00:08Z
ComputeHostsDeployment	INIT_COMPLETE	2016-11-24T07:00:08Z
RedisVirtualIP	CREATE_COMPLETE	2016-11-24T07:00:08Z
StorageVirtualIP	CREATE_COMPLETE	2016-11-24T07:00:08Z

오버클라우드 배포가 완료될 때까지 기다립니다.

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

- **IP 주소** - 오버클라우드에 액세스할 IP 주소입니다.
- **암호** - 오버클라우드의 **OpenStack admin** 사용자 암호입니다.

이 정보를 사용하여 오버클라우드에 액세스합니다.



Deployment succeeded

Stack CREATE completed successfully

Overcloud information:

- Overcloud IP address:
- Username: admin
- Password:

Delete Deployment

6.10. 오버클라우드 생성 완료

이렇게 하면 **director**의 UI를 통한 오버클라우드 생성이 완료됩니다. 생성 후 기능에 대한 자세한 내용은 **8 장. 오버클라우드 생성 후 작업 수행**을 참조하십시오.

7장. 사전 프로비저닝된 노드를 사용하여 기본 오버클라우드 구성

이 장에서는 **OpenStack Platform** 환경 구성에 사전 프로비저닝된 노드를 사용하기 위한 기본 구성 단계를 설명합니다. 이 시나리오는 여러 방식에서 표준 오버클라우드 생성 시나리오와 다릅니다.

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

이 시나리오는 사용자 정의 기능이 없는 기본 구성을 제공합니다. 하지만 고급 구성 옵션을 이 기본 오버클라우드에 추가하고 [고급 오버클라우드 사용자 정의](#) 가이드에 나와 있는 지침을 사용하여 사양에 맞게 사용자 정의를 할 수 있습니다.



중요

사전 프로비저닝된 노드를 오버클라우드의 **director**에서 프로비저닝된 노드와 혼합하는 기능은 지원되지 않습니다.

요구 사항

- [4장. 언더클라우드 설치](#)에서 생성한 **director** 노드.
- 노드의 베어 메탈 머신 그룹. 필요한 노드 수는 생성하려는 오버클라우드 유형에 따라 다릅니다 (오버클라우드 역할에 대한 자세한 내용은 [3.1절. “노드 배포 역할 플래닝”](#) 참조). 이러한 머신은 각 노드 유형에 대해 설정된 요구 사항에 부합해야 합니다. 이러한 요구 사항에 대한 자세한 내용은 [2.4절. “오버클라우드 요구 사항”](#)을 참조하십시오. 이러한 노드에는 **Red Hat Enterprise Linux 7.3** 운영 체제가 필요합니다.
- 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 **director**로 연결되는 3 계층(L3) 라우팅 가능한 네트워크 연결입니다. 이 시나리오의 예는 다음 IP 주소 할당을 사용합니다.

표 7.1. 프로비저닝 네트워크 IP 할당

노드 이름	IP 주소
Director	192.168.24.1
Controller	192.168.24.2

노드 이름	IP 주소
Compute	192.168.24.3

- 별도 네트워크 사용. **director**의 프로비저닝 네트워크가 라우팅 불가능한 개인 네트워크인 경우 서브넷에서 노드의 IP 주소를 정의하고 공용 API 엔드포인트를 통해 **director**와 통신할 수 있습니다. 이 시나리오에는 특정한 주의 사항이 있으며, 이 장에서는 [7.6절. “오버클라우드 노드에 별도의 네트워크 사용”](#)에서 나중에 검토합니다.
- 이 예에 있는 다른 모든 네트워크 유형에서도 **OpenStack** 서비스에 컨트롤 플레인 네트워크를 사용합니다. 하지만, 다른 네트워크 트래픽 유형에 대해 추가 네트워크를 생성할 수 있습니다.

7.1. 노드 구성을 위한 사용자 생성

이 프로세스의 후반 단계에서 **director**는 오버클라우드 노드에 **stack** 사용자로 SSH 액세스가 가능해야 합니다.

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

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

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

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

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

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

7.2. 노드의 운영 체제 등록

각 노드는 Red Hat 서브스크립션에 액세스할 수 있어야 합니다. 다음 절차는 각 노드를 Red Hat Content Delivery Network에 등록하는 방법을 보여줍니다. 각 노드에서 다음 단계를 수행하십시오.

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

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

2. Red Hat OpenStack Platform 11에 대한 인타이틀먼트 풀을 검색합니다.

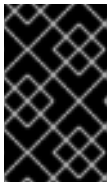
```
[root@controller ~]# sudo subscription-manager list --available --
all --matches="*OpenStack*"
```

3. 이전 단계에 있는 풀 ID를 사용하여 Red Hat OpenStack Platform 11 인타이틀먼트를 연결합니다.

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

4. 기본 리포지토리를 모두 비활성화한 다음 필수 Red Hat Enterprise Linux 리포지토리를 활성화합니다.

```
[root@controller ~]# sudo subscription-manager repos --disable=*
[root@controller ~]# sudo subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-7-server-extras-rpms --enable=rhel-7-server-rh-common-rpms --enable=rhel-ha-for-rhel-7-server-rpms --enable=rhel-7-server-openstack-11-rpms --enable=rhel-7-server-rhceph-2-osd-rpms --enable=rhel-7-server-rhceph-2-mon-rpms --enable=rhel-7-server-rhceph-2-tools-rpms
```



중요

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

5. 시스템을 업데이트하여 기본 시스템 패키지를 최신 상태로 합니다.

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

이제 노드에서 오버클라우드를 사용할 준비가 되었습니다.

7.3. 노드에 사용자 에이전트 설치

사전 프로비저닝된 각 노드는 **OpenStack Orchestration(heat)** 에이전트를 사용하여 **director**와 통신합니다. 각 노드의 에이전트는 **director**를 폴링하여 각 노드에 맞게 조정된 메타데이터를 가져옵니다. 이 메타데이터를 사용하여 에이전트가 각 노드를 구성할 수 있습니다.

각 노드에 오케스트레이션 초기 에이전트 패키지를 설치합니다.

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

7.4. DIRECTOR에 대한 SSL/TLS 액세스 구성

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

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

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

이렇게 하면 오버클라우드 노드에서 SSL/TLS를 통해 **director**의 공용 API에 액세스할 수 있습니다.

7.5. 컨트롤 플레인에 대한 네트워킹 구성

사전 프로비저닝된 오버클라우드 노드는 표준 HTTP 요청을 사용하여 **director**에서 메타데이터를 가져옵니다. 따라서 모든 오버클라우드 노드는 다음 중 하나에 액세스하려면 **L3** 권한이 필요합니다.

- **director**의 컨트롤 플레인 네트워크. **undercloud.conf** 파일의 **network_cidr** 매개 변수로 정의된 서브넷입니다. 노드에는 이 서브넷에 대한 직접 액세스 권한이나 서브넷으로 라우팅 가능한 액세스 권한이 필요합니다.
- **director**의 공용 API 엔드포인트. **undercloud.conf** 파일의 **undercloud_public_host** 매개 변수로 지정되어 있습니다. 이 옵션은 컨트롤 플레인에 대한 **L3** 경로가 없거나, 메타데이터에 대한 **director**를 폴링할 때 **SSL/TLS** 통신을 사용하려는 경우 사용할 수 있습니다. 공용 API 엔드포인트를 사용하도록 오버클라우드 노드를 구성하는 추가 설정 단계는 [7.6절. “오버클라우드 노드에 별도의 네트워크 사용”](#)을 참조하십시오.

director는 컨트롤 플레인 네트워크를 사용하여 표준 오버클라우드를 관리하고 구성합니다. 노드가 사전 프로비저닝된 오버클라우드의 경우 **director**가 사전 프로비저닝된 노드와 통신할 수 있도록 네트워크 구성을 일부 수정해야 할 수 있습니다.

네트워크 분리 사용

네트워크를 분리하여 컨트롤 플레인을 포함한 특정 네트워크를 사용하도록 서비스를 그룹화할 수 있습니다. [고급 오버클라우드 사용자 정의](#) 가이드에 여러 네트워크 분리 방법이 나와 있습니다. 또한 컨트롤 플레인에서 노드에 특정 IP 주소를 정의할 수 있습니다. 네트워크 분리 및 예측 가능한 노드 배치 방법에 대한 자세한 내용은 [고급 오버클라우드 사용자 정의](#) 가이드에 있는 다음 섹션을 참조하십시오.

- ["네트워크 분리"](#)
- ["노드 배치 제어"](#)



참고

네트워크 분리를 사용하는 경우 **NIC** 템플릿에 언더클라우드 액세스에 사용된 **NIC**가 포함되지 않도록 하십시오. 이러한 템플릿은 **NIC**를 재구성할 수 있으며, 이로 인해 배포 중에 연결 및 구성 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 구성해야 합니다. **director**의 프로비저닝 네트워크를 컨트롤 플레인으로 사용하는 경우 선택한 오버클라우드 IP 주소가 프로비저닝(**dhcp_start** 및 **dhcp_end**)과 **introspection(inspection_iprange)**에 대한 DHCP 범위를 벗어나지 않는지 확인합니다.

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

예측 가능한 IP 주소 할당 방법의 예는 다음 IP가 할당된 환경 파일을 사용합니다(**ctlplane-assignments.yaml**).

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

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

이 예에서 **OS::TripleO::DeployedServer::ControlPlanePort** 리소스는 매개변수 집합을 **director**에 전달하고, 사전 프로비저닝된 노드의 IP 할당을 정의합니다. **DeployedServerPortMap** 매개 변수는 각 오버클라우드 노드에 해당하는 IP 주소와 서브넷 CIDR을 정의합니다. 매핑은 다음을 정의합니다.

1. 할당 이름 - **<node_hostname>-<network>** 포맷을 따릅니다.
2. IP 할당 - 다음 매개 변수 패턴을 사용합니다.
 - **fixed_ips/ip_address** - 컨트롤 플레인의 고정 IP 주소를 정의합니다. 목록에 여러 개의 **ip_address** 매개 변수를 사용하여 여러 IP 주소를 정의합니다.
 - **subnets/cidr** - 서브넷의 CIDR 값을 정의합니다.

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

7.6. 오버클라우드 노드에 별도의 네트워크 사용

기본적으로 **director**는 프로비저닝 네트워크를 오버클라우드 컨트롤 플레인으로 사용합니다. 하지만 이 네트워크가 분리되어 라우팅이 불가능한 경우 구성 중에 노드에서 **director**의 내부 API와 통신할 수 없습니다. 이 경우 노드에 대해 별도의 네트워크를 정의하고, 공용 API로 **director**와 통신하도록 구성해야 할 수 있습니다.

이 시나리오에는 다음과 같은 여러 요구 사항이 있습니다.

- 오버클라우드 노드는 [7.5절. “컨트롤 플레인에 대한 네트워킹 구성”](#)의 기본 네트워크 구성을 사용할 수 있어야 합니다.
- 공용 API 엔드포인트 사용을 위해 **director**에서 SSL/TLS를 활성화해야 합니다. 자세한 내용은 [4.6절. “Director 구성”](#) 및 [부록 A. SSL/TLS 인증서 구성](#)을 참조하십시오.
- **director**에 대해 FQDN(정규화된 도메인 이름)을 정의해야 합니다. 이 FQDN은 **director**에 대해 라우팅 가능한 IP 주소를 확인할 수 있어야 합니다. **undercloud_public_host** 매개 변수를 **undercloud.conf** 파일에서 사용하여 이 FQDN을 설정합니다.

이 섹션의 예에서는 기본 시나리오와 다른 IP 주소 할당을 사용합니다.

표 7.2. 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	192.168.100.2
Compute	192.168.100.3

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

오케스트레이션 구성

director는 언더클라우드에서 SSL/TLS 통신을 활성화한 상태에서 대부분의 서비스에 공용 API 엔드포인트를 제공합니다. 하지만 **OpenStack Orchestration(heat)**은 내부 엔드포인트를 메타데이터에 대한 기본 공급자로 사용합니다. 따라서 오버클라우드 노드에서 공용 엔드포인트의 **OpenStack Orchestration**에 액세스할 수 있도록 언더클라우드에서 몇 가지를 수정해야 합니다. 이러한 수정에는 **director**에서의 일부 **Puppet hieradata** 변경이 포함됩니다.

undercloud.conf의 **hieradata_override**를 사용하면 언더클라우드 구성에 대한 추가 **Puppet hieradata**를 지정할 수 있습니다. **OpenStack Orchestration**과 관련된 **hieradata**를 수정하려면 다음 단계를 수행하십시오.

1. **hieradata_override** 파일을 아직 사용 중이 아닌 경우 새로 생성합니다. 이 예에서는 **/home/stack/hieradata.yaml**에 있는 파일을 사용합니다.
2. 다음 **hieradata**를 **/home/stack/hieradata.yaml**에 포함합니다.

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

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

3. **undercloud.conf**에서 **hieradata_override** 매개 변수를 **hieradata** 파일의 경로로 설정합니다.

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

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

이렇게 하면 오케스트레이션 메타데이터 서버가 **director**의 공용 API에서 URL을 사용하도록 전환됩니다.

IP 주소 할당

IP 할당 방법은 7.5절. “컨트롤 플레인에 대한 네트워킹 구성”과 비슷합니다. 하지만 컨트롤 플레인은 배포된 서버에서 라우팅되지 않으므로 컨트롤 플레인에 액세스할 가상 IP 주소를 포함하여 선택한 오버클라우드 노드 서브넷에서 IP 주소를 할당할 때 **DeployedServerPortMap** 매개 변수를 사용합니다. 다음은 7.5절. “컨트롤 플레인에 대한 네트워킹 구성”의 **ctlplane-assignments.yaml** 환경 파일에 대한 수정된

버전으로 이 네트워크 아키텍처를 사용합니다.

```
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 ❶

parameter_defaults:
  NeutronPublicInterface: eth1
  EC2MetadataIp: 192.168.100.1 ❷
  ControlPlaneDefaultRoute: 192.168.100.1
  DeployedServerPortMap:
    control_virtual_ip:
      fixed_ips:
        - ip_address: 192.168.100.1
      subnets:
        - cidr: 24
    controller0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.2
      subnets:
        - cidr: 24
    compute0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.3
      subnets:
        - cidr: 24
```

- ❶ **RedisVipPort** 리소스는 **network/ports/noop.yaml**에 매핑됩니다. 이 매핑은 기본 **Redis VIP** 주소가 컨트롤 플레인에서 만들어졌기 때문입니다. 이 경우 **noop**를 사용하여 이 컨트롤 플레인 매핑을 비활성화합니다.
- ❷ **EC2MetadataIp** 및 **ControlPlaneDefaultRoute** 매개 변수는 컨트롤 플레인 가상 IP 주소 값으로 설정됩니다. 기본 NIC 구성 템플릿에는 이러한 매개 변수가 필요하므로 ping할 수 있는 IP 주소를 사용하여 배포 중에 수행된 검증을 전달하도록 설정해야 합니다. 또는 이러한 매개 변수가 필요하지 않도록 NIC 구성을 사용자 정의합니다.

7.7. 사전 프로비저닝된 노드로 오버클라우드 생성

오버클라우드 배포에는 5.6절. “CLI 툴로 오버클라우드 생성”의 표준 CLI 메서드를 사용합니다. 사전 프로비저닝된 노드의 경우 배포 명령에 핵심 Heat 템플릿 컬렉션의 일부 추가 옵션 및 환경 파일이 필요합니다.

- **--disable-validations** - 사전 프로비저닝된 인프라에 사용되지 않는 서비스에 대한 기본 CLI 검증을 비활성화합니다.
- **environments/deployed-server-environment.yaml** - 사전 프로비저닝된 인프라를 생성 및 구성하는 기본 환경 파일입니다. 이 환경 파일은 **OS::Nova::Server** 리소스를 **OS::Heat::DeployedServer** 리소스로 대체합니다.

- **environments/deployed-server-bootstrap-environment-rhel.yaml** - 사전 프로비저닝된 서버에서 부트스트랩 스크립트를 실행할 환경 파일입니다. 이 스크립트는 추가 패키지를 설치하고 오버클라우드 노드에 기본 구성을 제공합니다.
- **environments/deployed-server-pacemaker-environment.yaml** - 사전 프로비저닝된 Controller 노드의 Pacemaker 구성에 대한 환경 파일입니다. 이 파일에 등록된 리소스에 대한 네임스페이스는 **deployed-server/deployed-server-roles-data.yaml**의 Controller 역할 이름을 사용하며, 기본값은 **ControllerDeployedServer**입니다.
- **deployed-server/deployed-server-roles-data.yaml** - 사용자 정의 역할에 대한 예제 파일입니다. 이 파일은 기본 **roles_data.yaml**을 복제 배포하지만, 각 역할에 대해 **disable_constraints: True** 매개 변수를 포함하기도 합니다. 이 매개 변수는 생성된 역할 템플릿에서 오케스트레이션 제약 조건을 비활성화합니다. 이러한 제약 조건은 사전 프로비저닝된 인프라에 사용되지 않은 서비스에 대한 것입니다. 고유한 사용자 정의 역할 파일을 사용하는 경우 **disable_constraints: True** 매개 변수를 각 역할과 함께 포함해야 합니다. 예를 들면 다음과 같습니다.

```
- name: ControllerDeployedServer
  disable_constraints: True
  CountDefault: 1
  ServicesDefault:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CephMon
    - OS::Triple0::Services::CephExternal
    - OS::Triple0::Services::CephRgw
    ...
```

다음은 사전 프로비저닝된 아키텍처에 고유한 환경 파일을 사용한 오버클라우드 배포 명령 예입니다.

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

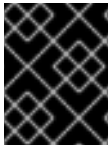
이 명령은 오버클라우드 구성을 시작합니다. 하지만 오버클라우드 노드 리소스가 **CREATE_IN_PROGRESS** 단계에 들어가면 배포 스택이 일시 중지됩니다.

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

이러한 일시 중지는 **director**가 오버클라우드 노드에서 오케스트레이션 에이전트가 메타데이터 서버를 폴링할 때까지 기다리기 때문에 발생합니다. 다음 섹션에는 메타데이터 서버 폴링을 시작할 노드를 구성하는 방법이 나와 있습니다.

7.8. 메타데이터 서버 폴링

이제 배포가 진행 중이지만, **CREATE_IN_PROGRESS** 단계에서 일시 중지됩니다. 다음 단계는 **director**에서 메타데이터 서버를 폴링하도록 오버클라우드 노드에서 오케스트레이션 에이전트를 구성하는 것입니다. 다음 두 가지 방법으로 이 작업을 수행할 수 있습니다.



중요

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

자동 설정

director의 핵심 Heat 템플릿 컬렉션에는 오버클라우드 노드에서 Heat 에이전트의 자동 설정을 수행하는 스크립트가 포함되어 있습니다. 이 스크립트를 사용하려면 **stack** 사용자로 **stackrc** 파일을 소싱하여 **director**에서 인증하고 오케스트레이션 서비스에 쿼리해야 합니다.

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

또한 스크립트에 노드 역할과 해당 IP 주소를 정의할 몇 가지 추가 환경 변수가 필요합니다. 이러한 환경 변수는 다음과 같습니다.

OVERCLOUD_ROLES

공백으로 구분된 구성할 역할 목록입니다. 이러한 역할은 역할 데이터 파일에 정의된 역할과 연관됩니다.

[ROLE]_hosts

각 역할에는 역할의 노드에 대해 공백으로 구분된 IP 주소 목록과 함께 환경 변수가 필요합니다.

다음 명령은 이러한 환경 변수를 설정하는 방법을 보여줍니다.

```
[stack@director ~]$ export OVERCLOUD_ROLES="ControllerDeployedServer
ComputeDeployedServer"
[stack@director ~]$ export ControllerDeployedServer_hosts="192.168.100.2"
[stack@director ~]$ export ComputeDeployedServer_hosts="192.168.100.3"
```

다음 스크립트를 실행하여 각 오버클라우드 노드에서 오케스트레이션 에이전트를 구성합니다.

```
[stack@director ~]$ /usr/share/openstack-tripleo-heat-templates/deployed-
server/scripts/get-occ-config.sh
```



참고

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

스크립트는 다음을 수행합니다.

- **director**의 오케스트레이션 서비스에서 각 노드에 대한 메타데이터 URL을 조회합니다.
- 노드에 액세스하고 특정 메타데이터 URL로 각 노드의 에이전트를 구성합니다.
- 오케스트레이션 에이전트 서비스를 다시 시작합니다.

스크립트가 완료되면 오버클라우드 노드가 **director**에서 오케스트레이션 서비스 폴링을 시작합니다. 스택 배포가 계속 진행됩니다.

수동 구성

사전 프로비저닝된 노드에서 오케스트레이션 에이전트를 수동으로 구성하려는 경우 다음 명령을 사용하여 **director**의 오케스트레이션 서비스에서 각 노드의 메타데이터 URL을 조회합니다.

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

다음은 각 노드에 대한 스택 이름과 메타데이터 URL을 표시한 것입니다.

```
== Metadata URL for ControllerDeployedServer 0 ==
http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-
edServer-ts6lr4tm5p44-deployed-server-td42md2tap4g/43d302fa-d4c2-40df-
b3ac-624d6075ef27?
temp_url_sig=58313e577a93de8f8d2367f8ce92dd7be7aac3a1&temp_url_expires=214
7483586

== Metadata URL for ComputeDeployedServer 0 ==
http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-
edServer-wdpk7upmz3eh-deployed-server-ghv7ptfikz2j/0a43e94b-fe02-427b-
9bfe-71d2b7bb3126?
temp_url_sig=8a50d8ed6502969f0063e79bb32592f4203a136e&temp_url_expires=214
7483586
```

각 오버클라우드 노드에서 다음을 수행합니다.

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

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

2. 해당 메타데이터 URL을 사용하도록 **/etc/os-collect-config.conf** 파일을 구성합니다. 예를 들면 **Controller** 노드에서 다음을 사용합니다.

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

[request]
metadata_url=http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7
125f05b764/ov-edServer-ts6lr4tm5p44-deployed-server-
td42md2tap4g/43d302fa-d4c2-40df-b3ac-624d6075ef27?
temp_url_sig=58313e577a93de8f8d2367f8ce92dd7be7aac3a1&temp_url_expir
es=2147483586
```

3. 파일을 저장합니다.

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

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

서비스를 구성하고 다시 시작하면 오케스트레이션 에이전트가 오버클라우드 구성에 대해 **director**의 오케스트레이션 서비스를 폴링합니다. 배포 스택은 해당 생성을 계속 진행하여 각 노드에 대한 스택이 **CREATE_COMPLETE**로 변경됩니다.

7.9. 오버클라우드 생성 모니터링

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

```
$ source ~/stackrc
$ heat stack-list --show-nested
```

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

7.10. 오버클라우드 액세스

director에서는 **director** 호스트에서 오버클라우드와 상호 작용을 설정하고 인증을 지원하는 스크립트를 생성합니다. **director**는 **overcloudrc** 파일을 **stack** 사용자의 홈 **director**에 저장합니다. 이 파일을 사용하려면 다음 명령을 실행합니다.

```
$ source ~/overcloudrc
```

이렇게 하면 **director** 노드에서 CLI의 오버클라우드와 상호 작용하는 데 필요한 환경 변수가 로드됩니다. **director** 노드와의 상호 작용으로 돌아가려면 다음을 실행합니다.

```
$ source ~/stackrc
```

7.11. 사전 프로비저닝된 노드 확장

사전 프로비저닝된 노드를 확장하는 프로세스는 [9장. 오버클라우드 확장](#)의 표준 확장 절차와 비슷합니다. 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 확장할 때 각 노드에서 **director**의 노드 수에 해당하도록 오케스트레이션 에이전트를 구성해야 합니다.

노드를 확장하는 일반적인 프로세스는 다음과 같습니다.

1. [요구 사항](#)에 따라 사전 프로비저닝된 새 노드를 준비합니다.
2. 노드를 확장합니다. 이러한 지침에 대해서는 [9장. 오버클라우드 확장](#)을 참조하십시오.

3. 배포 명령을 실행한 후 **director**가 새 노드 리소스를 생성할 때까지 기다립니다. [7.8절. “메타데이터 서버 폴링”](#)의 지침에 따라 **director**의 오케스트레이션 서버 메타데이터 URL을 폴링하도록 사전 프로비저닝된 노드를 수동으로 구성합니다.

사전 프로비저닝된 노드 축소

사전 프로비저닝된 노드가 있는 오버클라우드를 축소할 때 [9장. 오버클라우드 확장](#)에 표시된 대로 축소 지침을 따릅니다.

스택에서 오버클라우드 노드를 제거한 경우 이러한 노드의 전원을 끕니다. 표준 배포의 **director**에 있는 베어 메탈 서비스에서 이 기능을 제어합니다. 하지만 프로비저닝된 노드를 사용하는 경우 이러한 노드를 수동으로 종료하거나 각 물리적 시스템에 대해 전원 관리 컨트롤을 사용해야 합니다. 스택에서 노드를 제거한 후 노드의 전원을 끄지 않으면 작동 상태로 남아 있어 오버클라우드 환경의 일부로 재연결할 수 있습니다.

제거된 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 제거된 노드를 재사용하지 마십시오. 축소 프로세스는 오버클라우드 스택에서 노드를 제거만 하고 패키지를 제거하지는 않습니다.

7.12. 사전 프로비저닝된 오버클라우드 제거

사전 프로비저닝된 노드를 사용하는 전체 오버클라우드를 제거하면 표준 오버클라우드와 같은 절차를 사용합니다. 자세한 내용은 [8.11절. “오버클라우드 제거”](#)를 참조하십시오.

오버클라우드 제거 후 모든 노드의 전원을 끄고 기본 운영 체제 구성으로 다시 프로비저닝합니다.



참고

먼저 새로운 기본 운영 체제로 다시 프로비저닝하지 않고 오버클라우드에서 이전에 제거된 노드를 재사용하지 마십시오. 제거 프로세스는 오버클라우드 스택만 삭제하고 패키지를 제거하지는 않습니다.

7.13. 오버클라우드 생성 완료

이렇게 하면 사전 프로비저닝된 노드를 사용한 오버클라우드 생성이 완료됩니다. 생성 이후 기능에 대해서는 [8장. 오버클라우드 생성 후 작업 수행](#)을 참조하십시오.

8장. 오버클라우드 생성 후 작업 수행

이 장에서는 선택한 오버클라우드를 생성한 후 수행하는 기능 중 몇 가지에 대해 살펴봅니다.

8.1. 오버클라우드 테넌트 네트워크 생성

오버클라우드에는 인스턴스에 대한 테넌트 네트워크가 필요합니다. **overcloud**를 소싱하고 **Neutron**에 초기 테넌트 네트워크를 생성합니다. 예를 들면 다음과 같습니다.

```
$ source ~/overcloudrc
$ openstack network create default
$ openstack subnet create default --network default --gateway 172.20.1.1 -
-subnet-range 172.20.0.0/16
```

이렇게 하면 **default**라고 하는 기본 **Neutron** 네트워크가 생성됩니다. 오버클라우드에서 내부 DHCP 메커니즘을 사용하여 이 네트워크에서 IP 주소를 자동으로 할당합니다.

생성된 네트워크를 확인합니다.

```
$ openstack network list
+-----+-----+-----+
| id                | name      | subnets |
+-----+-----+-----+
| 95fadaa1-5dda-4777... | default   | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
+-----+-----+-----+
```

8.2. 오버클라우드 외부 네트워크 생성

유동 IP 주소를 인스턴스에 할당할 수 있도록 오버클라우드에서 외부 네트워크를 생성해야 합니다.

기본 VLAN 사용

이 절차에서는 외부 네트워크에 전용 인터페이스 또는 기본 VLAN이 설정되어 있다고 가정합니다.

overcloud에서 소스를 생성하고 **Neutron**에 외부 네트워크를 만듭니다. 예를 들면 다음과 같습니다.

```
$ source ~/overcloudrc
$ openstack network create public --external --provider-network-type flat
--provider-physical-network datacentre
$ 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 예에서 **public**이라는 이름으로 네트워크를 생성합니다. 오버클라우드에는 기본 유동 IP 풀에 대해 이러한 특정 이름이 필요합니다. 이는 [8.5절. “오버클라우드 검증”](#)의 검증 테스트에서도 중요합니다.

또한 이 명령은 네트워크를 **datacentre** 물리적 네트워크에 매핑합니다. 기본적으로 **datacentre**는 **br-ex** 브리지에 매핑됩니다. 오버클라우드 생성 중에 사용자 정의 **neutron** 설정을 사용하지 않은 경우 이 옵션을 기본값으로 둡니다.

기본이 아닌 VLAN 사용

기본 VLAN을 사용하지 않는 경우 다음 명령을 사용하여 네트워크를 VLAN에 할당합니다.

```
$ source ~/overcloudrc
$ openstack network create public --external --provider-network-type vlan
--provider-physical-network datacentre --provider-segment 104
$ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.1.51,end=10.1.1.250 --gateway 10.1.1.1 --subnet-range
10.1.1.0/24
```

provider:segmentation_id 값은 사용할 VLAN을 정의합니다. 이 경우 104를 사용할 수 있습니다.

생성된 네트워크를 확인합니다.

```
$ openstack network list
+-----+-----+-----+
| id                  | name          | subnets |
+-----+-----+-----+
| d474fe1f-222d-4e32... | public        | 01c5f621-1e0f-4b9d-9c30-7dc59592a52f |
+-----+-----+-----+
```

8.3. 추가 유동 IP 네트워크 생성

다음 조건을 충족하면 유동 IP 네트워크에서 **br-ex**가 아닌 모든 브리지를 사용할 수 있습니다.

- 네트워크 환경 파일에 **NeutronExternalNetworkBridge**가 **''**로 설정되어 있습니다.
- 배포 중에 추가 브리지를 매핑했습니다. 예를 들어 **br-floating**이라는 새 브리지를 **floating** 물리적 네트워크에 매핑하려면 다음 명령을 실행합니다.

```
$ source ~/stackrc
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml -e
~/templates/network-environment.yaml --neutron-bridge-mappings
datacentre:br-ex,floating:br-floating
```

오버클라우드 생성 후 유동 IP 네트워크를 생성합니다.

```
$ source ~/overcloudrc
$ openstack network create ext-net --external --provider-physical-network
floating --provider-network-type vlan --provider-segment 105
$ openstack subnet create ext-subnet --network ext-net --dhcp --
allocation-pool start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 --
subnet-range 10.1.2.0/24
```

8.4. 오버클라우드 공급자 네트워크 생성

공급자 네트워크는 배포된 오버클라우드 외부에 있는 네트워크에 물리적으로 연결된 네트워크입니다. 이 네트워크는 기존 인프라 네트워크이거나 유동 IP 대신 라우팅을 통해 인스턴스에 직접 외부 액세스를 제공하는 네트워크입니다.

공급자 네트워크를 생성할 때 브리지 매핑을 사용하는 물리적 네트워크와 연결합니다. 이는 유동 IP 네트워크 생성과 비슷합니다. **Compute** 노드가 VM 가상 네트워크 인터페이스를 연결된 네트워크 인터페이스에 직접 연결하므로, 공급자 네트워크를 **Controller**와 **Compute** 노드에 모두 추가합니다.

예를 들어 원하는 공급자 네트워크가 **br-ex** 브리지의 **VLAN**이면 다음 명령을 사용하여 **VLAN 201**의 공급자 네트워크를 추가합니다.

```
$ source ~/overcloudrc
$ openstack network create provider_network --provider-physical-network
  datacentre --provider-network-type vlan --provider-segment 201 --share
```

이 명령은 공유된 네트워크를 생성합니다. 또한 **--share**를 지정하지 않고 테넌트를 지정할 수 있습니다. 이 네트워크는 지정된 테넌트에서만 사용할 수 있습니다. 공급자 네트워크를 외부로 표시하는 경우 운영자만 해당 네트워크에 포트를 생성할 수 있습니다.

neutron에서 **DHCP** 서비스를 테넌트 인스턴스에 제공하도록 하려면 공급자 네트워크에 서브넷을 추가합니다.

```
$ source ~/overcloudrc
$ openstack subnet create provider-subnet --network provider_network --
  dhcp --allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway
  10.9.101.254 --subnet-range 10.9.101.0/24
```

다른 네트워크에서 공급자 네트워크를 통해 외부적으로 액세스해야 할 수 있습니다. 이 경우 다른 네트워크에서 공급자 네트워크를 통해 트래픽을 라우팅할 수 있도록 새 라우터를 생성합니다.

```
$ openstack router create external
$ openstack router set --external-gateway provider_network external
```

다른 네트워크를 이 라우터에 연결합니다. 예를 들어 **subnet1**이라는 서브넷이 있는 경우 다음 명령을 사용하여 해당 서브넷을 라우터에 연결할 수 있습니다.

```
$ openstack router add subnet external subnet1
```

이렇게 하면 **subnet1**이 라우팅 테이블에 추가되고, **subnet1**을 사용하는 트래픽이 공급자 네트워크로 라우팅됩니다.

8.5. 오버클라우드 검증

오버클라우드의 일련의 통합 테스트를 수행하도록 설정된 **OpenStack Integration Test Suite (tempest)** 툴셋을 사용합니다. 이 섹션에서는 통합 테스트 실행 준비에 대한 정보를 제공합니다. **OpenStack Integration Test Suite** 사용에 대한 전체 지침은 [OpenStack Integration Test Suite 가이드](#)를 참조하십시오.

Integration Test Suite를 실행하기 전

언더클라우드에서 이 테스트를 실행하는 경우 언더클라우드 호스트에서 오버클라우드의 내부 API 네트워크에 액세스할 수 있는지 확인합니다. 예를 들면 172.16.0.201/24 주소를 사용하여 내부 API 네트워크(ID: 201)에 액세스할 언더클라우드 호스트에 임시 VLAN을 추가합니다.

```
$ source ~/stackrc
$ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface
vlan201 type=internal
$ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev
vlan201
```

OpenStack Integration Test Suite를 실행하기 전에 오버클라우드에 **heat_stack_owner** 역할이 있는지 확인합니다.

```
$ source ~/overcloudrc
$ openstack role list
+-----+-----+
| ID                                           | Name           |
+-----+-----+
| 6226a517204846d1a26d15aae1af208f          | swiftoperator  |
| 7c7eb03955e545dd86bbfeb73692738b          | heat_stack_owner |
+-----+-----+
```

역할이 없는 경우 역할을 생성합니다.

```
$ openstack role create heat_stack_owner
```

Integration Test Suite를 실행한 후

검증을 완료한 후 오버클라우드의 내부 API에 대한 임시 연결을 제거합니다. 이 예에서는 다음 명령을 사용하여 이전에 생성한 VLAN을 언더클라우드에서 제거합니다.

```
$ source ~/stackrc
$ sudo ovs-vsctl del-port vlan201
```

8.6. 오버클라우드 환경 수정

다른 기능을 추가하도록 오버클라우드를 수정하거나 작동 방식을 변경할 수 있습니다. 오버클라우드를 수정하려면 사용자 정의 환경 파일 및 Heat 템플릿을 수정한 다음, 초기 오버클라우드 생성 시의 **openstack overcloud deploy** 명령을 다시 실행합니다. 예를 들어 [5.6절. “CLI 툴로 오버클라우드 생성”](#)을 사용하여 오버클라우드를 생성한 경우 다음 명령을 다시 실행합니다.

```
$ source ~/stackrc
$ 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. 그러면 오버클라우드가 다시 생성되지는 않지만, 기존 오버클라우드가 변경됩니다.

새 환경 파일을 포함하려면 해당 파일을 **-e** 옵션과 함께 **openstack overcloud deploy** 명령에 추가합니다. 예를 들면 다음과 같습니다.

```
$ source ~/stackrc
$ 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
```

이렇게 하면 환경 파일의 새 매개 변수와 리소스가 스택에 포함됩니다.



중요

오버클라우드의 구성을 수동으로 수정하지 않는 것이 좋습니다. **director**가 나중에 이러한 수정 사항을 덮어쓸 수 있습니다.

8.7. 가상 머신을 오버클라우드로 가져오기

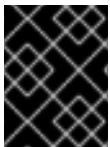
기존 **OpenStack** 환경이 있으며 해당 가상 머신을 **Red Hat OpenStack Platform** 환경으로 마이그레이션하려는 경우 다음 절차를 사용하십시오.

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
```

오버클라우드에 내보낸 이미지를 업로드하고 새 인스턴스를 실행합니다.

```
$ openstack image create imported_image --file exported_vm.qcow2 --disk-
format qcow2 --container-format bare
$ openstack server create imported_instance --key-name default --flavor
m1.demo --image imported_image --nic net-id=net_id
```



중요

각 VM 디스크를 기존 **OpenStack** 환경에서 새 **Red Hat OpenStack Platform**으로 복사합니다. **QCOW**를 사용한 스냅샷은 원래 계층 시스템이 손실됩니다.

8.8. 오버클라우드 **COMPUTE** 노드에서 **VM** 마이그레이션

상황에 따라 오버클라우드 **Compute** 노드에서 유지보수를 수행해야 할 수 있습니다. 다운 타임을 방지하려면 **Compute** 노드에서 **VM**을 오버클라우드의 다른 **Compute** 노드로 마이그레이션합니다.

director는 안전한 마이그레이션을 제공하기 위해 모든 **Compute** 노드를 구성합니다. 모든 **Compute** 노드에는 마이그레이션 프로세스 중에 다른 **Compute** 노드에 대한 액세스 권한을 각 호스트의 **nova** 사용자에게 제공하기 위해 공유 **SSH** 키도 필요합니다. **director**는 **OS::TripleO::Services::NovaCompute** 구성 가능한 서비스를 사용하여 이 키를 생성합니다. 구성 가능한 이 서비스는 기본적으로 모든 **Compute** 역할에 포함된 기본 서비스 중 하나입니다([고급 오버클라우드 사용자 정의의 "구성 가능한 서비스 및 사용자 정의 역할"](#) 참조).

인스턴스를 마이그레이션하려면 다음을 수행합니다:

1. 언더클라우드에서 재부팅할 **Compute** 노드를 선택하여 비활성화합니다.

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
```

2. **Compute** 노드에 모든 인스턴스를 나열합니다.

```
$ openstack server list --host [hostname] --all-projects
```

3. 비활성화된 호스트에서 각 인스턴스를 마이그레이션합니다. 다음 명령 중 하나를 사용합니다.

- a. 인스턴스를 선택한 특정 호스트로 마이그레이션합니다.

```
$ openstack server migrate [instance-id] --live [target-host] --wait
```

- b. **nova-scheduler**에서 대상 호스트를 자동으로 선택하도록 합니다.

```
$ nova live-migration [instance-id]
```



참고

nova 명령으로 인해 몇 가지 사용 중단 경고가 표시될 수 있으며, 이러한 경고는 무시해도 됩니다.

4. 마이그레이션이 완료될 때까지 기다립니다.
5. 인스턴스가 **Compute** 노드에서 마이그레이션되었는지 확인합니다.

```
$ openstack server list --host [hostname] --all-projects
```

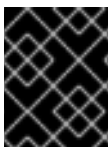
6. **Compute** 노드에서 모든 인스턴스를 마이그레이션할 때까지 이 단계를 반복합니다.

이렇게 하면 **Compute** 노드의 모든 인스턴스가 마이그레이션됩니다. 이제 인스턴스 중단 없이 노드에서 유지보수를 수행할 수 있습니다. **Compute** 노드를 활성화된 상태로 되돌리려면 다음 명령을 실행합니다.

```
$ source ~/overcloudrc
$ openstack compute service set [hostname] nova-compute --enable
```

8.9. ANSIBLE 자동화 실행

director에서는 OpenStack Platform 환경에서 **Ansible** 기반 자동화를 실행하는 기능을 제공합니다. **director**는 **tripleo-ansible-inventory** 명령을 사용하여 환경에 노드의 동적 인벤토리를 생성합니다.



중요

동적 인벤토리 톨은 언더클라우드와 기본 **controller** 및 **compute** 오버클라우드 노드만 포함합니다. 다른 역할은 지원되지 않습니다.

노드의 동적 인벤토리를 보려면 **stackrc**를 소싱한 후 **tripleo-ansible-inventory** 명령을 실행합니다.

```
$ source ~/stackrc
$ tripleo-ansible-inventory --list
```

--list 옵션은 모든 호스트에 대한 세부 사항을 제공합니다.

이렇게 하면 동적 인벤토리가 JSON 형식으로 출력됩니다.

```
{
  "overcloud": {
    "children": ["controller", "compute"],
    "vars": {
      "ansible_ssh_user": "heat-admin"
    },
    "controller": ["192.168.24.2"],
    "undercloud": {
      "hosts": ["localhost"],
      "vars": {
        "overcloud_horizon_url": "http://192.168.24.4:80/dashboard",
        "overcloud_admin_password": "abcdefghijklmnopqrstuvwxyz12345678",
        "ansible_connection": "local"
      },
      "compute": ["192.168.24.3"]
    }
  }
}
```

현재 환경에서 Ansible 플레이북을 실행하려면 **ansible** 명령을 실행하고 **-i** 옵션을 사용하여 동적 인벤토리 톨의 전체 경로를 포함합니다. 예를 들면 다음과 같습니다.

```
ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER OPTIONS]
```

- **[HOSTS]**를 사용할 호스트 유형으로 변경합니다. 예를 들면 다음과 같습니다.
 - 모든 Controller 노드인 경우 **controller**
 - 모든 Compute 노드인 경우 **compute**
 - 모든 오버클라우드 하위 노드인 경우 **overcloud**. 예: **controller** 및 **compute**
 - 언더클라우드인 경우 **undercloud**
 - 모든 노드인 경우 **"*"**
- **[OTHER OPTIONS]**를 추가 Ansible 옵션으로 변경합니다. 몇 가지 유용한 옵션은 다음과 같습니다.
 - **--ssh-extra-args='-o StrictHostKeyChecking=no'**: 호스트 키 검사에서 확인을 바이패스합니다.
 - **-u [USER]**: Ansible 자동화를 실행하는 SSH 사용자를 변경합니다. 오버클라우드에 대한 기본 SSH 사용자는 동적 인벤토리에서 **ansible_ssh_user** 매개 변수를 사용하여 자동으로 정의됩니다. **-u** 옵션은 이 매개 변수를 재정의합니다.
 - **-m [MODULE]**: 특정 Ansible 모듈을 사용합니다. 기본값은 Linux 명령을 실행하는 **command**입니다.
 - **-a [MODULE_ARGS]**: 선택한 모듈에 대한 인수를 정의합니다.



중요

오버클라우드에서 Ansible 자동화는 표준 오버클라우드 스택 외부에 속합니다. 즉, 후속 **openstack overcloud deploy** 명령을 실행하면 오버클라우드 노드에서 OpenStack Platform 서비스에 대한 Ansible 기반 구성을 덮어쓸 수 있습니다.

8.10. 오버클라우드 삭제 방지

heat stack-delete overcloud 명령 사용으로 오버클라우드가 실수로 제거되지 않도록 하기 위해 Heat에 특정 작업을 제한할 일련의 정책이 포함되어 있습니다. **/etc/heat/policy.json**을 편집하고 다음 매개 변수를 검색합니다.

```
"stacks:delete": "rule:deny_stack_user"
```

다음과 같이 변경합니다.

```
"stacks:delete": "rule:deny_everybody"
```

파일을 저장합니다.

이렇게 하면 오버클라우드가 **heat** 클라이언트와 함께 제거되지 않습니다. 오버클라우드 제거를 허용하려면 정책을 원래 값으로 되돌립니다.

8.11. 오버클라우드 제거

원하는 경우 전체 오버클라우드를 제거할 수 있습니다.

기존 오버클라우드를 삭제합니다.

```
$ source ~/stackrc
$ openstack overcloud delete overcloud
```

오버클라우드 삭제를 확인합니다.

```
$ openstack stack list
```

삭제에는 몇 분 정도 시간이 걸립니다.

제거가 완료되면 배포 시나리오의 표준 단계에 따라 오버클라우드를 다시 생성합니다.

9장. 오버클라우드 확장



주의

Compute 인스턴스에 고가용성을 사용하면(또는 **Compute** 인스턴스에 대한 고가용성에 설명된 대로 인스턴스 HA) 업그레이드 또는 확장 작업이 불가능합니다. 이 작업을 시도하면 모두 실패합니다.

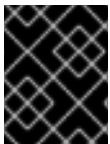
인스턴스 HA를 활성화한 경우 업그레이드 또는 확장을 수행하기 전에 비활성화합니다. 이를 위해 [롤백](#)에 설명된 대로 **롤백**을 수행합니다.

오버클라우드 생성 후에 노드를 추가하거나 제거해야 할 필요가 있을 수 있습니다. 예를 들면 오버클라우드에 더 많은 **Compute** 노드를 추가해야 할 수 있습니다. 이러한 경우 오버클라우드를 업데이트해야 합니다.

아래 표를 사용하여 각 노드 유형의 확장 지원 여부를 확인하십시오.

표 9.1. 각 노드 유형의 확장 지원

노드 유형	확장 가능 여부	축소 가능 여부	비고
Controller	N	N	
Compute	Y	Y	
Ceph Storage 노드	Y	N	초기 오버클라우드 생성 시 적어도 하나의 Ceph Storage 노드가 있어야 합니다.
Block Storage 노드	N	N	
Object Storage 노드	Y	Y	수동으로 링을 관리해야 함(9.6절, “ Object Storage 노드 교체 ” 참조)



중요

오버클라우드를 확장하려면 적어도 **10GB**의 사용 가능한 공간이 있어야 합니다. 이 공간은 노드 프로비저닝 프로세스 중에 이미지 변환 및 캐싱에 사용됩니다.

9.1. 기타 노드 추가

director의 노드 풀에 더 많은 노드를 추가하려면 등록할 새 노드 세부 사항이 포함된 새 **JSON** 파일(예: **newnodes.json**)을 생성합니다.

```
{
```

```

"nodes": [
  {
    "mac": [
      "dd:dd:dd:dd:dd:dd"
    ],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "pxe_ipmitool",
    "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": "pxe_ipmitool",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.208"
  }
]
}

```

이러한 매개 변수에 대한 설명은 [5.1절. “오버클라우드에 노드 등록”](#)을 참조하십시오.

다음 명령을 실행하여 이러한 노드를 등록합니다.

```
$ openstack baremetal import --json newnodes.json
```

새 노드를 등록한 후에 이러한 노드에 대한 **introspection** 프로세스를 시작합니다. 새로운 각 노드에 대해 다음 명령을 사용합니다.

```

$ openstack baremetal node manage [NODE UUID]
$ openstack overcloud node introspect [NODE UUID] --provide

```

이 경우 노드의 하드웨어 속성을 감지하여 벤치마킹합니다.

introspection 프로세스가 완료되면 해당 역할에 대해 각각의 새 노드를 태깅합니다. 예를 들어 **Compute** 노드의 경우 다음 명령을 사용합니다.

```

$ openstack baremetal node set --property
capabilities='profile:compute,boot_option:local' [NODE UUID]

```

배포 중에 사용할 부팅 이미지를 설정합니다. **bm-deploy-kernel** 및 **bm-deploy-ramdisk** 이미지의 UUID를 확인합니다.

```

$ openstack image list
+-----+-----+-----+

```

ID	Name
09b40e3d-0382-4925-a356-3a4b4f36b514	bm-deploy-kernel
765a46af-4417-4592-91e5-a300ead3faf6	bm-deploy-ramdisk
ef793cd0-e65c-456a-a675-63cd57610bd5	overcloud-full
9a51a6cb-4670-40de-b64b-b70f4dd44152	overcloud-full-initrd
4f7e33f4-d617-47c1-b36f-cbe90f132e5d	overcloud-full-vmlinuz

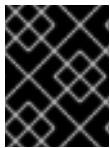
새 노드의 **deploy_kernel** 및 **deploy_ramdisk** 설정에 대해 이러한 UUID를 설정합니다.

```
$ openstack baremetal node set --driver-info deploy_kernel='09b40e3d-0382-4925-a356-3a4b4f36b514' [NODE UUID]
$ openstack baremetal node set --driver-info deploy_ramdisk='765a46af-4417-4592-91e5-a300ead3faf6' [NODE UUID]
```

오버클라우드를 확장하려면 역할에 필요한 노드 수를 지정하고 **openstack overcloud deploy**를 다시 실행해야 합니다. 예를 들어 5개의 **Compute** 노드로 확장하려면 다음을 실행합니다.

```
$ openstack overcloud deploy --templates --compute-scale 5 [OTHER_OPTIONS]
```

이렇게 하면 전체 오버클라우드 스택이 업데이트됩니다. 이 경우 스택만 업데이트됩니다. 오버클라우드가 삭제되고 스택이 교체되지는 않습니다.



중요

초기 오버클라우드 생성 시의 모든 환경 파일과 옵션을 포함합니다. 여기에는 **Compute** 이외 노드에 대한 동일한 확장 매개 변수가 포함됩니다.

9.2. COMPUTE 노드 제거

오버클라우드에서 **Compute** 노드를 제거해야 하는 경우가 있을 수 있습니다. 예를 들면 문제가 있는 **Compute** 노드를 교체해야 할 수 있습니다.



중요

오버클라우드에서 **Compute** 노드를 제거하기 전에 노드에서 다른 **Compute** 노드로 워크로드를 마이그레이션합니다. 자세한 내용은 [8.8절. “오버클라우드 Compute 노드에서 VM 마이그레이션”](#)을 참조하십시오.

다음으로 오버클라우드에서 노드의 **Compute** 서비스를 비활성화합니다. 그러면 노드에서 새 인스턴스가 예약되지 않습니다.

```
$ source ~/stack/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
$ source ~/stack/stackrc
```

오버클라우드 노드를 제거하려면 로컬 템플릿 파일을 사용하여 **director**에서 **overcloud** 스택으로 업데이트해야 합니다. 먼저, 오버클라우드 스택의 **UUID**를 확인합니다.

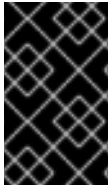
```
$ openstack stack list
```

삭제할 노드의 **UUID**를 확인합니다.

```
$ openstack server list
```

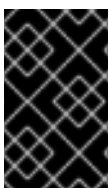
다음 명령을 실행하여 스택에서 노드를 삭제하고 그에 따라 플랜을 업데이트합니다.

```
$ openstack overcloud node delete --stack [STACK_UUID] --templates -e  
[ENVIRONMENT_FILE] [NODE1_UUID] [NODE2_UUID] [NODE3_UUID]
```



중요

오버클라우드를 생성할 때 추가 환경 파일을 전달한 경우 **-e** 또는 **--environment-file** 옵션을 사용하여 오버클라우드를 불필요하게 수동으로 변경하지 않도록 환경 파일을 다시 지정합니다.



중요

작업을 계속하려면 **openstack overcloud node delete** 명령을 실행하여 완료합니다. **openstack stack list** 명령을 사용하여 **overcloud** 스택이 **UPDATE_COMPLETE** 상태로 전환했는지 확인합니다.

마지막으로 노드의 **Compute** 서비스를 제거합니다.

```
$ source ~/stack/overcloudrc  
$ openstack compute service list  
$ openstack compute service delete [service-id]  
$ source ~/stack/stackrc
```

노드의 **Open vSwitch** 에이전트를 제거합니다.

```
$ source ~/stack/overcloudrc  
$ openstack network agent list  
$ openstack network agent delete [openvswitch-agent-id]  
$ source ~/stack/stackrc
```

이제 오버클라우드에서 노드를 제거하여 다른 용도로 노드를 다시 프로비저닝할 수 있습니다.

9.3. COMPUTE 노드 교체

Compute 노드에 장애가 발생하면 해당 노드를 작동하는 노드로 교체할 수 있습니다. **Compute** 노드 교체에는 다음 프로세스를 사용합니다.

- 기존 **Compute** 노드에서 워크로드를 마이그레이션하고 해당 노드를 종료합니다. 이 프로세스에 대해서는 [8.8절. “오버클라우드 Compute 노드에서 VM 마이그레이션”](#)을 참조하십시오.
- 오버클라우드에서 **Compute** 노드를 제거합니다. 이 프로세스에 대해서는 [9.2절. “Compute 노드 제거”](#)를 참조하십시오.
- 오버클라우드를 새 **Compute** 노드로 확장합니다. 이 프로세스에 대해서는 [9.1절. “기타 노드 추가”](#)를 참조하십시오.

이 프로세스를 수행하면 인스턴스의 가용성에 영향을 주지 않고 노드를 교체할 수 있습니다.

9.4. CONTROLLER 노드 교체

특정 상황에서 고가용성 클러스터의 **Controller** 노드에 장애가 발생할 수 있습니다. 이러한 경우 클러스터에서 해당 노드를 제거하고 새 **Controller** 노드로 교체해야 합니다. 또한 노드가 클러스터의 다른 노드에 연결되도록 해야 합니다.

이 섹션에서는 **Controller** 노드를 교체하는 방법에 대해 설명합니다. 프로세스에는 **Controller** 노드 교체에 대한 요청으로 오버클라우드를 업데이트하는 **openstack overcloud deploy** 명령을 실행하는 작업이 포함됩니다. 이 프로세스는 완전히 자동으로 수행되지 않습니다. 오버클라우드 스택 업데이트 프로세스 중에 **openstack overcloud deploy** 명령은 특정 시점에서 오류를 보고하고 오버클라우드 스택 업데이트를 중지합니다. 이 시점에서 프로세스를 수행하려면 일부 수동 조작이 필요합니다. 그런 다음 **openstack overcloud deploy** 프로세스를 계속할 수 있습니다.



중요

다음 절차는 고가용성 환경에만 적용됩니다. **Controller** 노드를 하나만 사용하는 경우 이 절차를 사용하지 마십시오.

9.4.1. 사전 점검

오버클라우드 **Controller** 노드를 교체하기 전에 Red Hat OpenStack Platform 환경의 현재 상태를 확인하는 것이 중요합니다. 현재 상태를 확인하면 **Controller** 교체 프로세스 중에 복잡한 문제가 발생하는 것을 방지할 수 있습니다. 다음 사전 점검 목록을 사용하여 **Controller** 노드 교체를 수행하는 것이 안전한지 확인합니다. 언더클라우드에서 검사 명령을 모두 실행합니다.

1. 언더클라우드에서 **overcloud** 스택의 현재 상태를 확인합니다.

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

overcloud 스택 및 해당 하위 스택에 **CREATE_COMPLETE** 또는 **UPDATE_COMPLETE**가 있어야 합니다.

2. 언더클라우드 데이터베이스 백업을 수행합니다.

```
$ mkdir /home/stack/backup
$ sudo mysqldump --all-databases --quick --single-transaction | gzip
> /home/stack/backup/dump_db_undercloud.sql.gz
```

3. 새 노드를 프로비저닝할 때 언더클라우드에 이미지 캐싱 및 변환을 수행하는 데 필요한 10GB의 사용 가능한 스토리지가 있는지 확인합니다.
4. 실행 중인 **Controller** 노드에서 **Pacemaker**의 상태를 확인합니다. 예를 들어 192.168.0.47이 실행 중인 **Controller** 노드의 IP 주소인 경우 다음 명령을 사용하여 **Pacemaker** 상태 정보를 가져옵니다.

```
$ ssh heat-admin@192.168.0.47 'sudo pcs status'
```

출력에 기존 노드에서 실행 중인 서비스와 실패한 노드에서 중지된 모든 서비스가 표시됩니다.

5. 오버클라우드의 MariaDB 클러스터에 있는 각 노드에서 다음 매개 변수를 확인합니다.

- **wsrep_local_state_comment**: Synced
- **wsrep_cluster_size**: 2

다음 명령을 사용하여 실행 중인 각 **Controller** 노드에서 이러한 매개 변수를 확인합니다(각 IP 주소에 **192.168.0.47** 및 **192.168.0.46** 사용).

```
$ for i in 192.168.0.47 192.168.0.46 ; do echo "**** $i ****" ; ssh
heat-admin@$i "sudo mysql --exec=\"SHOW STATUS LIKE
'wsrep_local_state_comment'\" ; sudo mysql --exec=\"SHOW STATUS
LIKE 'wsrep_cluster_size'\""; done
```

6. **RabbitMQ** 상태를 확인합니다. 예를 들어 **192.168.0.47**이 실행 중인 **Controller** 노드의 IP 주소이면 다음 명령을 사용하여 상태 정보를 가져옵니다.

```
$ ssh heat-admin@192.168.0.47 "sudo rabbitmqctl cluster_status"
```

running_nodes 키는 사용 가능한 두 개의 노드만 표시하고, 실패한 노드는 표시하지 않습니다.

7. 펜싱이 활성화된 경우 비활성화합니다. 예를 들어 **192.168.0.47**이 실행 중인 **Controller** 노드의 IP 주소이면 다음 명령을 사용하여 펜싱을 비활성화합니다.

```
$ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-
enabled=false"
```

다음 명령을 사용하여 펜싱 상태를 확인합니다.

```
$ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-
enabled"
```

8. **director** 노드에서 **nova-compute** 서비스를 확인합니다.

```
$ sudo systemctl status openstack-nova-compute
$ openstack hypervisor list
```

출력에 모든 유지보수 이외의 모드 노드가 **up**으로 표시됩니다.

9. 모든 언더클라우드 서비스가 실행 중인지 확인합니다.

```
$ sudo systemctl -t service
```

9.4.2. 노드 교체

제거할 노드의 인덱스를 확인합니다. 노드 인덱스는 **nova list** 출력의 인스턴스 이름에서 접미사입니다.

```
[stack@director ~]$ openstack server list
+-----+-----+
| ID                                     | Name                               |
+-----+-----+
| 861408be-4027-4f53-87a6-cd3cf206ba7a | overcloud-compute-0              |
| 0966e9ae-f553-447a-9929-c4232432f718 | overcloud-compute-1              |
| 9c08fa65-b38c-4b2e-bd47-33870bfff06c7 | overcloud-compute-2              |
| a7f0f5e1-e7ce-4513-ad2b-81146bc8c5af | overcloud-controller-0           |
| cfefaf60-8311-4bc3-9416-6a824a40a9ae | overcloud-controller-1           |
| 97a055d4-aefd-481c-82b7-4a5f384036d2 | overcloud-controller-2           |
+-----+-----+
```

이 예에서는 **overcloud-controller-1** 노드를 제거하고 **overcloud-controller-3**으로 교체합니다. 먼저, **director**가 실패한 노드를 다시 프로비저닝하지 않도록 노드를 유지보수 모드로 설정합니다. **nova list**의 인스턴스 ID를 **openstack baremetal node list**의 노드 ID와 상호 연결합니다.

```
[stack@director ~]$ openstack baremetal node list
+-----+-----+-----+-----+
| UUID                                     | Name | Instance UUID |
+-----+-----+-----+-----+
| 36404147-7c8a-41e6-8c72-a6e90afc7584 | None | 7bee57cf-4a58-4eaf-b851-2a8bf6620e48 |
| 91eb9ac5-7d52-453c-a017-c0e3d823efd0 | None | None |
| 75b25e9a-948d-424a-9b3b-f0ef70a6eacf | None | None |
| 038727da-6a5c-425f-bd45-fda2f4bd145b | None | 763bfec2-9354-466a-ae65-2401c13e07e5 |
| dc2292e6-4056-46e0-8848-d6e96df1f55d | None | 2017b481-706f-44e1-852a-2ee857c303c4 |
| c7eadcea-e377-4392-9fc3-cf2b02b7ec29 | None | 5f73c7d7-4826-49a5-b6be-8bfd558f3b41 |
| da3a8d19-8a59-4e9d-923a-6a336fe10284 | None | cfefaf60-8311-4bc3-9416-6a824a40a9ae |
| 807cb6ce-6b94-4cd1-9969-5c47560c2eee | None | c07c13e6-a845-4791-9628-260110829c3a |
+-----+-----+-----+-----+
```

노드를 유지보수 모드로 설정합니다.

```
[stack@director ~]$ openstack baremetal node maintenance set da3a8d19-8a59-4e9d-923a-6a336fe10284
```

새 노드를 **control** 프로파일로 태깅합니다.

```
[stack@director ~]$ openstack baremetal node set --property capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-9b3b-f0ef70a6eacf
```

오버클라우드의 데이터베이스는 교체 절차 중에 계속 실행되고 있어야 합니다. 이 절차 중에 **Pacemaker**에서 **Galera**를 중지하지 않도록 하려면 실행 중인 **Controller** 노드를 선택하고, 언더클라우드에서 **Controller** 노드의 IP 주소를 사용하여 다음 명령을 실행합니다.

```
[stack@director ~]$ ssh heat-admin@192.168.0.47 "sudo pcs resource unmanage galera"
```

제거할 노드 인덱스를 정의하는 YAML 파일(**~/templates/remove-controller.yaml**)을 생성합니다.

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['1']}]
```




참고

Corosync에서 설정 시도 횟수를 줄임으로써 교체 프로세스 속도를 높일 수 있습니다. `~/templates/remove-controller.yaml` 환경 파일에서 **CorosyncSettleTries** 매개 변수를 지정합니다.

```
parameter_defaults:
  CorosyncSettleTries: 5
```

노드 인덱스를 식별한 후 오버클라우드를 재배포하고 **remove-controller.yaml** 환경 파일을 추가합니다.

```
[stack@director ~]$ openstack overcloud deploy --templates --control-scale
3 -e ~/templates/remove-controller.yaml [OTHER OPTIONS]
```

오버클라우드를 생성할 때 추가 환경 파일이나 옵션을 전달한 경우 오버클라우드가 예기치 않게 변경되지 않도록 여기서 다시 전달합니다.

하지만 **-e ~/templates/remove-controller.yaml**은 이 인스턴스에서 한 번만 필요합니다.

director에서 기존 노드를 제거하고, 새 노드를 생성한 후 오버클라우드 스택을 업데이트합니다. 다음 명령을 사용하여 오버클라우드 스택의 상태를 확인할 수 있습니다.

```
[stack@director ~]$ openstack stack list --nested
```

9.4.3. 수동 조작

ControllerNodesPostDeployment 단계 중에 **ControllerDeployment_Step1**의

UPDATE_FAILED 오류로 오버클라우드 스택 업데이트가 중지됩니다. 이는 일부 **Puppet** 모듈이 노드 교체를 지원하지 않기 때문입니다. 프로세스의 이 시점에는 일부 수동 조작이 필요합니다. 다음 구성 단계를 따르십시오.

1. **Controller** 노드의 IP 주소 목록을 가져옵니다. 예를 들면 다음과 같습니다.

```
[stack@director ~]$ openstack server list
... +-----+ ... +-----+
... | 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   |
... +-----+ ... +-----+
```

2. 기존 노드의 `/etc/corosync/corosync.conf` 파일에서 제거된 노드의 **nodeid** 값을 확인합니다. 예를 들어 기존 노드는 **192.168.0.47**에서 **overcloud-controller-0**입니다.

```
[stack@director ~]$ ssh heat-admin@192.168.0.47 "sudo cat
/etc/corosync/corosync.conf"
```

이렇게 하면 제거된 노드(**overcloud-controller-1**)의 ID가 포함된 **nodelist**가 표시됩니다.

■

```

    nodelist {
      node {
        ring0_addr: overcloud-controller-0
        nodeid: 1
      }
      node {
        ring0_addr: overcloud-controller-1
        nodeid: 2
      }
      node {
        ring0_addr: overcloud-controller-2
        nodeid: 3
      }
    }
  }
}

```

나중에 사용할 수 있도록 제거된 노드의 **nodeid** 값을 기록해 둡니다. 이 예에서는 **2**입니다.

3. 각 노드의 **Corosync** 구성에서 장애가 발생한 노드를 삭제하고 **Corosync**를 다시 시작합니다. 이 예에서는 **overcloud-controller-0** 및 **overcloud-controller-2**에 로그인하고 다음 명령을 실행합니다.

```

[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster
reload corosync"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster
reload corosync"

```

4. 나머지 노드 중 하나에 로그인하고 **crm_node** 명령을 사용하여 클러스터에서 노드를 삭제합니다.

```

[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ sudo crm_node -R overcloud-
controller-1 --force

```

이 노드에 로그인한 상태를 유지합니다.

5. **RabbitMQ** 클러스터에서 장애가 발생한 노드를 삭제합니다.

```

[heat-admin@overcloud-controller-0 ~]$ sudo rabbitmqctl
forget_cluster_node rabbit@overcloud-controller-1

```

6. **MongoDB**에서 장애가 발생한 노드를 삭제합니다. 먼저, 노드의 내부 **API** 연결을 위한 **IP** 주소를 확인합니다.

```

[heat-admin@overcloud-controller-0 ~]$ sudo netstat -tulnp | grep
27017
tcp          0      0 192.168.0.47:27017    0.0.0.0:*
LISTEN      13415/mongod

```

노드가 **primary** 복제 세트인지 확인합니다.

```
[root@overcloud-controller-0 ~]# echo "db.isMaster()" | mongo --host
192.168.0.47:27017
MongoDB shell version: 2.6.11
connecting to: 192.168.0.47:27017/echo
{
  "setName" : "tripleo",
  "setVersion" : 1,
  "ismaster" : true,
  "secondary" : false,
  "hosts" : [
    "192.168.0.47:27017",
    "192.168.0.46:27017",
    "192.168.0.45:27017"
  ],
  "primary" : "192.168.0.47:27017",
  "me" : "192.168.0.47:27017",
  "electionId" : ObjectId("575919933ea8637676159d28"),
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 1000,
  "localTime" : ISODate("2016-06-09T09:02:43.340Z"),
  "maxWireVersion" : 2,
  "minWireVersion" : 0,
  "ok" : 1
}
bye
```

이는 현재 노드가 주 노드인지 표시합니다. 주 노드가 아닌 경우 **primary** 키에 표시된 노드의 IP 주소를 사용합니다.

주 노드에서 MongoDB에 연결합니다.

```
[heat-admin@overcloud-controller-0 ~]$ mongo --host 192.168.0.47
MongoDB shell version: 2.6.9
connecting to: 192.168.0.47:27017/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
tripleo:PRIMARY>
```

MongoDB 클러스터의 상태를 확인합니다.

```
tripleo:PRIMARY> rs.status()
```

_id 키를 사용하여 노드를 식별하고 **name** 키를 사용하여 장애가 발생한 노드를 제거합니다. 이 경우에는 **name**에 **192.168.0.45:27017** 이 있는 노드 1을 제거합니다.

```
tripleo:PRIMARY> rs.remove('192.168.0.45:27017')
```



중요

PRIMARY 복제 세트에 대해 이 명령을 실행해야 합니다.

```
"replSetReconfig command must be sent to the current replica set primary."
```

위척 메시지가 표시되는 경우 **PRIMARY**로 지정된 노드에서 **MongoDB**에 다시 로그인합니다.



참고

장애가 발생한 노드의 복제 세트를 제거할 때 일반적으로 다음과 같이 출력됩니다.

```
2016-05-07T03:57:19.541+0000 DBClientCursor::init call() failed
2016-05-07T03:57:19.543+0000 Error: error doing query: failed at src/mongo/shell/query.js:81
2016-05-07T03:57:19.545+0000 trying reconnect to 192.168.0.47:27017 (192.168.0.47) failed
2016-05-07T03:57:19.547+0000 reconnect 192.168.0.47:27017 (192.168.0.47) ok
```

MongoDB를 종료합니다.

```
tripleo:PRIMARY> exit
```

7. Galera 클러스터의 노드 목록을 업데이트합니다.

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource update galera wsrep_cluster_address=gcomm://overcloud-controller-0,overcloud-controller-3,overcloud-controller-2
```

8. 새 노드에서 Galera 클러스터를 검사하도록 구성합니다. 기존 노드에서 새 노드의 동일한 위치로 **/etc/sysconfig/clustercheck**를 복사합니다.
9. 새 노드에서 **root** 사용자의 Galera 액세스를 구성합니다. 기존 노드에서 새 노드의 동일한 위치에 **/root/.my.cnf**를 복사합니다.
10. 새 노드를 클러스터에 추가합니다.

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster node add overcloud-controller-3
```

11. 각 노드에서 **/etc/corosync/corosync.conf** 파일을 확인합니다. 새 노드의 **nodeid**가 제거된 노드와 같은 경우 값을 새 **nodeid** 값으로 업데이트합니다. 예를 들어 **/etc/corosync/corosync.conf** 파일에 새 노드에 대한 항목이 포함되어 있습니다 (**overcloud-controller-3**).

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
```

```

    }
    node {
      ring0_addr: overcloud-controller-2
      nodeid: 3
    }
    node {
      ring0_addr: overcloud-controller-3
      nodeid: 2
    }
  }
}

```

이 예에서 새 노드는 제거된 노드의 같은 **nodeid**를 사용합니다. 이 값을 사용되지 않은 노드 ID 값으로 업데이트합니다. 예를 들면 다음과 같습니다.

```

node {
  ring0_addr: overcloud-controller-3
  nodeid: 4
}

```

새 노드를 포함하여 각 **Controller** 노드의 **/etc/corosync/corosync.conf** 파일에 이 **nodeid** 값을 업데이트합니다.

- 기존 노드에서만 **Corosync** 서비스를 다시 시작합니다. 예를 들면 **overcloud-controller-0**에서 다음을 수행합니다.

```

[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster reload
corosync

```

overcloud-controller-2에서 다음을 수행합니다.

```

[heat-admin@overcloud-controller-2 ~]$ sudo pcs cluster reload
corosync

```

새 노드에서 이 명령을 실행하지 마십시오.

- 새 **Controller** 노드를 시작합니다.

```

[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster start
overcloud-controller-3

```

- Galera** 클러스터를 다시 시작하고 **Pacemaker** 관리로 되돌아갑니다.

```

[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource cleanup
galera
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource manage
galera

```

- Pacemaker**를 통해 일부 서비스를 활성화하고 다시 시작합니다. 클러스터는 현재 유지보수 모드에 있으므로 서비스를 활성화하려면 일시적으로 비활성화해야 합니다. 예를 들면 다음과 같습니다.

```

[heat-admin@overcloud-controller-3 ~]$ sudo pcs property set
maintenance-mode=false --wait

```

16. Galera 서비스가 모든 노드에서 시작될 때까지 기다립니다.

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep galera
-A1
Master/Slave Set: galera-master [galera]
Masters: [ overcloud-controller-0 overcloud-controller-2 overcloud-controller-3 ]
```

필요한 경우 새 노드에서 **cleanup**을 수행합니다.

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs resource cleanup
galera --node overcloud-controller-3
```

17. 클러스터를 다시 유지보수 모드로 전환합니다.

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs property set
maintenance-mode=true --wait
```

수동 구성이 완료되었는지 확인합니다. 오버클라우드 배포 명령을 다시 실행하여 스택 업데이트를 계속합니다.

```
[stack@director ~]$ openstack overcloud deploy --templates --control-scale
3 [OTHER OPTIONS]
```



중요

오버클라우드를 생성할 때 추가 환경 파일이나 옵션을 전달한 경우 오버클라우드를 예기치 않게 변경하지 않도록 여기서 다시 전달하십시오. 하지만 **remove-controller.yaml** 파일은 더 이상 필요하지 않습니다.

9.4.4. 오버클라우드 서비스 구성 완료

오버클라우드 스택 업데이트가 완료되면 일부 최종 설정이 필요합니다. **Controller** 노드 중 하나에 로그인하고 **Pacemaker**에서 중지된 서비스를 새로 고칩니다.

```
[heat-admin@overcloud-controller-0 ~]$ for i in `sudo pcs status|grep -B2
Stop |grep -v "Stop\|Start"|awk -F"[" ' /\[/ {print
substr($NF,0,length($NF)-1)}'`; do echo $i; sudo pcs resource cleanup $i;
done
```

최종 스택 확인을 수행하여 서비스가 올바르게 실행 중인지 확인합니다.

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



참고

서비스가 실패한 경우 **pcs resource cleanup** 명령을 사용하여 문제를 해결한 후 서비스를 다시 시작합니다.

director를 종료합니다.

```
[heat-admin@overcloud-controller-0 ~]$ exit
```

9.4.5. L3 에이전트 라우터 호스팅 구성 완료

오버클라우드와 상호 작용할 수 있도록 **overcloudrc** 파일을 소싱합니다. L3 에이전트가 오버클라우드 환경의 라우터를 제대로 호스팅하는지 라우터에서 확인합니다. 이 예에서는 이름 **r1**에 라우터를 사용합니다.

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

이 목록에 새 노드 대신 기존 노드가 여전히 표시될 수 있습니다. 기존 노드를 교체하려면 환경에 L3 네트워크 에이전트를 나열합니다.

```
[stack@director ~]$ neutron agent-list | grep "neutron-l3-agent"
```

새 노드와 기존 노드에서 에이전트의 **UUID**를 식별합니다. 라우터를 새 노드의 에이전트에 추가하고 기존 노드에서 라우터를 제거합니다. 예를 들면 다음과 같습니다.

```
[stack@director ~]$ neutron l3-agent-router-add fd6b3d6e-7d8c-4e1a-831a-4ec1c9ebb965 r1
[stack@director ~]$ neutron l3-agent-router-remove b40020af-c6dd-4f7a-b426-eba7bac9dbc2 r1
```

라우터에서 최종 확인을 수행하고 모두 활성 상태인지 확인합니다.

```
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

이전 **Controller** 노드를 가리키는 기존 **Neutron** 에이전트를 삭제합니다. 예를 들면 다음과 같습니다.

```
[stack@director ~]$ neutron agent-list -F id -F host | grep overcloud-controller-1
| ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb | overcloud-controller-1.localdomain |
[stack@director ~]$ neutron agent-delete ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb
```

9.4.6. Compute 서비스 구성 완료

제거된 노드의 **Compute** 서비스가 여전히 오버클라우드에 있으므로 제거해야 합니다. 오버클라우드와 상호 작용할 수 있도록 **overcloudrc** 파일을 소싱합니다. **Compute** 서비스에서 제거된 노드가 있는지 확인합니다.

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ nova service-list | grep "overcloud-controller-1.localdomain"
```

노드의 **Compute** 서비스를 제거합니다. 예를 들어 **overcloud-controller-1.localdomain**에 대한 **nova-scheduler** 서비스의 ID가 5이면 다음 명령을 실행합니다.

```
[stack@director ~]$ nova service-delete 5
```

제거된 노드의 각 서비스에 대해 이 작업을 수행합니다.

새 노드에서 **openstack-nova-consoleauth** 서비스를 확인합니다.

```
[stack@director ~]$ nova service-list | grep consoleauth
```

서비스가 실행 중이 아닌 경우 **Controller** 노드에 로그인하고 서비스를 다시 시작합니다.

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ pcs resource restart openstack-
nova-consoleauth
```

9.4.7. 결론

장애가 발생한 **Controller** 노드 및 해당 관련 서비스가 이제 새 노드로 교체됩니다.



중요

9.6절. “Object Storage 노드 교체”에서처럼 **Object Storage**에서 링 파일 자동 빌드를 비활성화한 경우, 새 노드에 대해 **Object Storage** 링 파일을 수동으로 빌드해야 합니다. 링 파일을 수동으로 빌드하는 작업에 대한 자세한 내용은 **9.6절. “Object Storage 노드 교체”**를 참조하십시오.

9.5. CEPH STORAGE 노드 교체

director에서는 **director**에서 생성한 클러스터에 있는 **Ceph Storage** 노드를 교체할 수 있습니다. [오버클라우드 용 Red Hat Ceph Storage](#)에서 자세한 내용을 확인하십시오.

9.6. OBJECT STORAGE 노드 교체

이 섹션에서는 클러스터의 무결성을 유지보수하는 동안 **Object Storage** 노드를 교체하는 방법을 설명합니다. 이 예에서는 두 개의 노드로 구성된 **Object Storage** 클러스터에서 **overcloud-objectstorage-1** 노드를 교체해야 합니다. 이 절차에서는 한 개의 노드를 더 추가한 다음 **overcloud-objectstorage-1**을 제거하는 것을 목표로 합니다(효율적으로 교체).

1. 다음 콘텐츠가 포함된 **~/templates/swift-upscale.yaml**이라는 환경 파일을 생성합니다.

```
parameter_defaults:
  ObjectStorageCount: 3
```

ObjectStorageCount는 환경에 있는 **Object Storage** 노드 수를 정의합니다. 이 경우 2개에서 3개 노드로 확장합니다.

2. **swift-upscale.yaml** 파일을 오버클라우드의 나머지 환경 파일(**ENVIRONMENT_FILES**)에 **openstack overcloud deploy**의 일부로 추가합니다.

```
$ openstack overcloud deploy --templates ENVIRONMENT_FILES -e swift-
upscale.yaml
```




참고

해당 매개 변수가 이전 환경 파일 매개 변수 보다 우선이 되도록 **swift-upscale.yaml**을 환경 파일 목록 끝에 추가합니다.

재배포가 완료되면 이제 오버클라우드에 추가 **Object Storage** 노드가 포함됩니다.

- 이제 데이터를 새 노드에 복제해야 합니다. 노드를 제거하기 전에(이 경우 **overcloud-objectstorage-1**) *replication pass*가 새 노드에서 완료될 때까지 대기해야 합니다. **/var/log/swift/swift.log**에서 복제 전달 진행 상황을 확인할 수 있습니다. 전달이 완료되면 **Object Storage** 서비스는 다음과 같은 항목을 로그에 기록합니다.

```
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
```

- 링에서 이전 노드를 제거하려면 이전 링을 생략하도록 **swift-upscale.yaml**에서 **ObjectStorageCount**를 줄입니다. 이 경우에는 2로 줄입니다.

```
parameter_defaults:
  ObjectStorageCount: 2
```

- remove-object-node.yaml**이라는 새 환경 파일을 생성합니다. 이 파일은 지정된 **Object Storage** 노드를 식별하고 제거합니다. 다음 콘텐츠는 **overcloud-objectstorage-1**을 제거하도록 지정합니다.

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

- 배포 명령에 두 환경 파일을 지정합니다.

```
$ openstack overcloud deploy --templates ENVIRONMENT_FILES -e swift-upscale.yaml -e remove-object-node.yaml ...
```

director가 오버클라우드에서 **Object Storage** 노드를 삭제하고 노드 제거를 적용하도록 오버클라우드에서 나머지 노드를 업데이트합니다.

10장. 노드 재부팅

상황에 따라 언더클라우드와 오버클라우드에서 노드를 재부팅해야 합니다. 다음 절차는 다른 노드 유형을 재부팅하는 방법을 보여줍니다. 다음 참고 사항을 확인하십시오.

- 한 역할에 있는 모든 노드를 재부팅하는 경우 각 노드를 개별적으로 재부팅하는 것이 좋습니다. 그러면 재부팅 중에 해당 역할에 대한 서비스를 유지하는 데 도움이 됩니다.
- **OpenStack Platform** 환경의 노드를 모두 재부팅하는 경우 다음 목록을 사용하여 재부팅 순서를 지정합니다.

권장되는 노드 재부팅 순서

1. **director** 재부팅
2. **Controller** 노드 재부팅
3. **Ceph Storage** 노드 재부팅
4. **Compute** 노드 재부팅
5. **Object Storage** 노드 재부팅

10.1. DIRECTOR 재부팅

director 노드를 재부팅하려면 다음 프로세스를 따르십시오.

1. 노드를 재부팅합니다.

```
$ sudo reboot
```

2. 노드가 부팅될 때까지 기다립니다.

노드가 부팅되면 모든 서비스의 상태를 확인합니다.

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch"
```



참고

재부팅 후 **openstack-nova-compute**가 활성화될 때까지 약 10분 정도 걸릴 수 있습니다.

오버클라우드 및 해당 노드가 있는지 확인합니다.

```
$ source ~/stackrc
$ openstack server list
$ openstack baremetal node list
$ openstack stack list
```

10.2. CONTROLLER 노드 재부팅

Controller 노드를 재부팅하려면 다음 프로세스를 따르십시오.

1. 재부팅할 노드를 선택합니다. 노드에 로그인하고 재부팅합니다.

```
$ sudo reboot
```

클러스터에서 나머지 **Controller** 노드는 재부팅 중에 고가용성 서비스가 유지됩니다.

2. 노드가 부팅될 때까지 기다립니다.
3. 노드에 로그인하고 클러스터 상태를 확인합니다.

```
$ sudo pcs status
```

노드가 클러스터에 다시 연결됩니다.



참고

재부팅 후 서비스가 실패하면 **sudo pcs resource cleanup**을 실행합니다. 그러면 오류가 제거되고 각 리소스의 상태가 **Started**로 설정됩니다. 오류가 지속되면 Red Hat에 연락하여 지원을 요청하십시오.

4. Controller 노드의 모든 **systemd** 서비스가 활성 상태인지 확인합니다.

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch"
```

5. 노드에서 로그아웃하고 재부팅할 다음 **Controller** 노드를 선택한 후 모든 **Controller** 노드가 재부팅될 때까지 이 절차를 반복합니다.

10.3. CEPH STORAGE 노드 재부팅

Ceph Storage 노드를 재부팅하려면 다음 절차를 따르십시오.

1. Ceph MON 또는 Controller 노드에 로그인하고 Ceph Storage 클러스터 재조정을 일시적으로 비활성화합니다.

```
$ sudo ceph osd set noout
$ sudo ceph osd set norebalance
```

2. 재부팅할 첫 번째 Ceph Storage 노드를 선택하고 로그인합니다.
3. 노드를 재부팅합니다.

```
$ sudo reboot
```

4. 노드가 부팅될 때까지 기다립니다.
5. 노드에 로그인하고 클러스터 상태를 확인합니다.

```
$ sudo ceph -s
```

pgmap이 모든 **pgs**를 정상적 (**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 **Controller** 노드에 로그인하고 클러스터 재조정을 다시 활성화합니다.

```
$ sudo ceph osd unset noout
$ sudo ceph osd unset norebalance
```

8. 최종 상태 검사를 수행하여 클러스터가 **HEALTH_OK**를 보고하는지 확인합니다.

```
$ sudo ceph status
```

10.4. COMPUTE 노드 재부팅

각 **Compute** 노드를 개별적으로 재부팅하고 **OpenStack Platform** 환경의 인스턴스 다운타임이 제로가 되도록 합니다. 다음 워크플로우에 따라 실행합니다.

1. 재부팅할 **Compute** 노드 선택
2. 해당 인스턴스를 다른 **Compute** 노드로 마이그레이션
3. 빈 **Compute** 노드 재부팅

모든 **Compute** 노드 및 해당 **UUID**를 나열합니다.

```
$ nova list | grep "compute"
```

재부팅할 **Compute** 노드를 선택하고 다음 프로세스를 사용하여 먼저 해당 인스턴스를 마이그레이션합니다.

1. 언더클라우드에서 재부팅할 **Compute** 노드를 선택하여 비활성화합니다.

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
```

2. **Compute** 노드에 모든 인스턴스를 나열합니다.

```
$ openstack server list --host [hostname] --all-projects
```

3. 비활성화된 호스트에서 각 인스턴스를 마이그레이션합니다. 다음 명령 중 하나를 사용합니다.

- a. 인스턴스를 선택한 특정 호스트로 마이그레이션합니다.

```
$ openstack server migrate [instance-id] --live [target-host]--wait
```

- b. **nova-scheduler**에서 대상 호스트를 자동으로 선택하도록 합니다.

```
$ nova live-migration [instance-id]
```



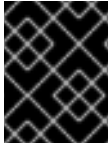
참고

nova 명령으로 인해 몇 가지 사용 중단 경고가 표시될 수 있으며, 이러한 경고는 무시해도 됩니다.

4. 마이그레이션이 완료될 때까지 기다립니다.
5. 인스턴스가 **Compute** 노드에서 마이그레이션되었는지 확인합니다.

```
$ openstack server list --host [hostname] --all-projects
```

6. **Compute** 노드에서 모든 인스턴스를 마이그레이션할 때까지 이 단계를 반복합니다.



중요

인스턴스 구성 및 마이그레이션에 대한 전체 지침은 [8.8절. “오버클라우드 Compute 노드에서 VM 마이그레이션”](#)을 참조하십시오.

다음 프로세스를 사용하여 **Compute** 노드를 재부팅합니다.

1. **Compute** 노드에 로그인하고 재부팅합니다.

```
$ sudo reboot
```

2. 노드가 부팅될 때까지 기다립니다.

3. **Compute** 노드를 다시 활성화합니다.

```
$ source ~/overcloudrc
$ openstack compute service set [hostname] nova-compute --enable
```

4. **Compute** 노드가 활성화되었는지 확인합니다.

```
$ openstack compute service list
```

10.5. OBJECT STORAGE 노드 재부팅

Object Storage 노드를 재부팅하려면 다음 프로세스를 따르십시오.

1. 재부팅할 **Object Storage** 노드를 선택합니다. 이 노드에 로그인하고 재부팅합니다.

```
$ sudo reboot
```

2. 노드가 부팅될 때까지 기다립니다.

3. 노드에 로그인하고 상태를 확인합니다.

```
$ sudo systemctl list-units "openstack-swift*"
```

4. 노드에서 로그아웃한 후 다음 **Object Storage** 노드에서 이 프로세스를 반복합니다.

11장. DIRECTOR 문제 해결

director 프로세스의 특정 단계에서 오류가 발생할 수 있습니다. 이 섹션에서는 일반적인 문제에 대한 몇 가지 진단 정보를 제공합니다.

director 구성 요소의 일반 로그를 참조하십시오.

- **/var/log** 디렉터리에는 여러 **OpenStack Platform** 구성 요소에 대한 로그와 표준 **Red Hat Enterprise Linux** 애플리케이션에 대한 로그가 포함되어 있습니다.
- **journal** 서비스는 여러 구성 요소에 대한 로그를 제공합니다. **ironic**은 **openstack-ironic-api** 및 **openstack-ironic-conductor** 유닛을 제공합니다. 마찬가지로 **ironic-inspector**도 **openstack-ironic-inspector** 및 **openstack-ironic-inspector-dnsmasq** 유닛을 사용합니다. 각각의 해당 구성 요소에 두 개 유닛 모두를 사용하십시오. 예를 들면 다음과 같습니다.

```
$ sudo journalctl -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq
```

- **ironic-inspector**는 또한 램디스크 로그를 **/var/log/ironic-inspector/ramdisk/**에 **gz**로 압축된 **tar** 파일로 저장합니다. 파일 이름에는 노드의 날짜, 시간 및 **IPMI** 주소가 포함됩니다. **introspection** 문제 진단에 이러한 로그를 사용하십시오.

11.1. 노드 등록 문제 해결

노드 등록 문제는 일반적으로 잘못된 노드 세부 사항 문제로 인해 발생합니다. 이 경우 **ironic**을 사용하여 등록된 노드 데이터 문제를 해결합니다. 다음은 몇 가지 예입니다.

할당된 포트 **UUID**를 확인합니다:

```
$ ironic node-port-list [NODE UUID]
```

MAC 주소를 업데이트합니다:

```
$ ironic port-update [PORT UUID] replace address=[NEW MAC]
```

다음 명령을 실행합니다:

```
$ ironic node-update [NODE UUID] replace driver_info/ipmi_address=[NEW IPMI ADDRESS]
```

11.2. 하드웨어 INTROSPECTION 문제 해결

introspection 프로세스 실행은 끝까지 완료되어야 합니다. 하지만 검색 램디스크가 응답하지 않을 경우 **ironic**의 검색 데몬(**ironic-inspector**)이 1시간(기본값) 후에 시간 초과됩니다. 경우에 따라 이는 검색 램디스크의 버그가 원인이 되는 경우도 있지만, 일반적으로는 **BIOS** 부팅 설정 등 잘못된 환경 구성으로 인해 설정 오류가 발생합니다.

다음은 잘못된 환경 구성이 발생하는 몇 가지 일반적인 시나리오 및 진단/해결 방법에 대한 설명입니다.

시작 노드 Introspection의 오류

일반적으로 **introspection** 프로세스는 **ironic** 서비스에 대한 복합 명령 역할을 하는 **baremetal**

introspection을 사용합니다. 하지만 **ironic-inspector**으로 **introspection**을 직접 실행하는 경우 **AVAILABLE** 상태의 노드를 검색하지 못할 수 있습니다(해당 상태는 검색이 아니라 배포에 대한 상태를 의미하기 때문). 검색 전에 노드 상태를 **MANAGEABLE** 상태로 변경하십시오.

```
$ ironic node-set-provision-state [NODE UUID] manage
```

그런 다음 검색이 완료되면 프로비저닝 전에 다시 **AVAILABLE**로 변경합니다.

```
$ ironic node-set-provision-state [NODE UUID] provide
```

검색 프로세스 중지

introspection 프로세스를 중지합니다.

```
$ openstack baremetal introspection abort [NODE UUID]
```

프로세스가 시간 초과될 때까지 기다릴 수 있습니다. 필요한 경우 **timeout** 설정을 **/etc/ironic-inspector/inspector.conf**에서 다른 기간(분)으로 변경합니다.

Introspection 램디스크 할당

introspection 램디스크는 동적 로그인 요소를 사용합니다. 즉, **introspection** 디버깅 중에 노드에 액세스 할 임시 암호 또는 **SSH** 키를 제공할 수 있습니다. 램디스크 액세스를 설정하려면 다음 프로세스를 사용하십시오.

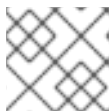
1. **openssl passwd -1** 명령에 임시 암호를 제공하여 **MD5** 해시를 생성합니다. 예를 들면 다음과 같습니다.

```
$ openssl passwd -1 mytestpassword
$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
```

2. **/httpboot/inspector.ipxe** 파일을 편집하고, **kernel**로 시작하는 행을 찾은 다음 **rootpwd** 매개 변수 및 **MD5** 해시를 첨부합니다. 예를 들면 다음과 같습니다.

```
kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-hardware,logs
systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1
ipa-inspection-benchmarks=cpu,mem,disk
rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
```

또는 공용 **SSH** 키와 함께 **sshkey** 매개 변수를 첨부할 수 있습니다.



참고

rootpwd와 **sshkey** 매개 변수에 모두 따옴표가 필요합니다.

3. **introspection**을 시작하고 **arp** 명령 또는 **DHCP** 로그에서 **IP** 주소를 찾습니다.

```
$ arp
$ sudo journalctl -u openstack-ironic-inspector-dnsmasq
```

4. 임시 암호 또는 **SSH** 키를 사용하여 **root** 사용자로 **SSH** 연결합니다.

```
$ ssh root@192.168.24.105
```

Introspection 스토리지 확인

director는 OpenStack Object Storage(swift)를 사용하여 introspection 프로세스 중에 가져온 하드웨어 데이터를 저장합니다. 이 서비스가 실행 중이 아니면 introspection이 실패할 수 있습니다. OpenStack Object Storage와 관련된 모든 서비스를 확인하여 서비스가 실행 중인지 확인합니다.

```
$ sudo systemctl list-units openstack-swift*
```

11.3. 워크플로우 및 실행 문제 해결

OpenStack Workflow(mistral) 서비스는 여러 OpenStack 작업을 워크플로우에 그룹화합니다. Red Hat OpenStack Platform은 이러한 워크플로우 집합을 사용하여 CLI 및 웹 UI에 일반적인 기능을 수행합니다. 여기에는 베어 메탈 노드 제어, 검증 및 플랜 관리 및 오버클라우드 배포가 포함됩니다.

예를 들어 **openstack overcloud deploy** 명령을 실행할 때 OpenStack Workflow 서비스는 두 개의 워크플로우를 실행합니다. 첫 번째는 배포 플랜을 업로드합니다.

```
Removing the current plan files
Uploading new plan files
Started Mistral Workflow. Execution ID: aef1e8c6-a862-42de-8bce-
073744ed5e6b
Plan updated
```

두 번째는 오버클라우드 배포를 시작합니다.

```
Deploying templates in the directory /tmp/tripleoclient-LhRlHX/tripleo-
heat-templates
Started Mistral Workflow. Execution ID: 97b64abe-d8fc-414a-837a-
1380631c764d
2016-11-28 06:29:26Z [overcloud]: CREATE_IN_PROGRESS Stack CREATE started
2016-11-28 06:29:26Z [overcloud.Networks]: CREATE_IN_PROGRESS state
changed
2016-11-28 06:29:26Z [overcloud.HeatAuthEncryptionKey]: CREATE_IN_PROGRESS
state changed
2016-11-28 06:29:26Z [overcloud.ServiceNetMap]: CREATE_IN_PROGRESS state
changed
...
```

워크플로우 오브젝트

OpenStack 워크플로우는 다음 오브젝트를 사용하여 워크플로우를 추적합니다.

동작

관련 작업이 실행되면 OpenStack에서 수행하는 특정한 명령입니다. 예를 들면 셸 스크립트 실행 또는 HTTP 요청 수행이 있습니다. 일부 OpenStack 구성 요소에는 OpenStack 워크플로우에서 사용하는 기본 제공 동작이 있습니다.

작업

실행할 동작 및 해당 동작을 실행한 결과를 정의합니다. 이러한 작업에는 일반적으로 동작 및 그러한 동작과 연관된 다른 워크플로우가 있습니다. 작업이 완료되면 작업의 성공 또는 실패 여부에 따라 워크플로우에서 다른 작업을 지시합니다.

워크플로우

함께 그룹화되고 특정 순서로 실행되는 작업 집합입니다.

실행

특정 동작, 작업 또는 워크플로우 실행을 정의합니다.

워크플로우 오류 진단

OpenStack Workflow는 지속적으로 실행 기록을 취합할 수 있으므로 특정 명령이 실패했을 경우 문제를 식별할 수 있습니다. 예를 들어 워크플로우 실행이 실패하는 경우 문제 지점을 확인할 수 있습니다. 실패한 상태 **ERROR**가 있는 워크플로우 실행을 표시합니다.

```
$ mistral execution-list | grep "ERROR"
```

실패한 워크플로우 실행의 UUID를 가져와서(예: 3c87a885-0d37-4af8-a471-1b392264a7f5) 실행 및 해당 출력을 표시합니다.

```
$ mistral execution-get 3c87a885-0d37-4af8-a471-1b392264a7f5
$ mistral execution-get-output 3c87a885-0d37-4af8-a471-1b392264a7f5
```

이는 실행에서 실패한 작업에 대한 정보를 제공합니다. **mistral execution-get**은 또한 실행에 사용된 워크플로우를 표시합니다(예: **tripleo.plan_management.v1.update_deployment_plan**). 다음 명령을 사용하면 전체 워크플로우 정의를 볼 수 있습니다.

```
$ mistral execution-get-definition
tripleo.plan_management.v1.update_deployment_plan
```

이 명령은 워크플로우에서 특정 작업이 발생한 위치를 식별하는 데 유용합니다.

비슷한 명령 구문을 사용하여 작업 실행 및 해당 결과를 볼 수도 있습니다.

```
$ mistral action-execution-list
$ mistral action-execution-get b59245bf-7183-4fcf-9508-c83ec1a26908
$ mistral action-execution-get-output b59245bf-7183-4fcf-9508-c83ec1a26908
```

이는 문제의 원인이 되는 특정 작업을 식별하는 데 유용합니다.

11.4. 오버클라우드 생성 문제 해결

배포가 실패할 수 있는 세 가지 계층이 있습니다.

- 오케스트레이션 (heat 및 nova 서비스)
- 베어 메탈 프로비저닝 (ironic 서비스)
- 배포 후 구성(Puppet)

오버클라우드 배포가 이러한 수준에서 실패한 경우 OpenStack 클라이언트 및 서비스 로그 파일을 사용하여 실패한 배포를 진단합니다.

11.4.1. 오케스트레이션

대부분의 경우 Heat는 오버클라우드 생성 실패 후에 실패한 오버클라우드 스택을 표시합니다.

```
$ heat stack-list
```

```

+-----+-----+-----+-----+
+-----+
| id                | stack_name | stack_status      | creation_time
|
+-----+-----+-----+-----+
+-----+
| 7e88af95-535c-4a55... | overcloud  | CREATE_FAILED     | 2015-04-
06T17:57:16Z |
+-----+-----+-----+-----+
+-----+

```

스택 목록이 비어 있으면 초기 **Heat** 설정에 문제가 있음을 나타냅니다. **Heat** 템플릿 및 구성 옵션을 확인하고, **openstack overcloud deploy** 실행 후 표시된 오류 메시지를 확인합니다.

11.4.2. 베어 메탈 프로비저닝

ironic에서 등록된 모든 노드와 노드의 현재 상태를 확인합니다.

```

$ ironic node-list

+-----+-----+-----+-----+-----+-----+
+-----+
| UUID      | Name | Instance UUID | Power State | Provision State |
Maintenance |
+-----+-----+-----+-----+-----+-----+
+-----+
| f1e261... | None | None          | power off   | available        | False
|
| f0b8c1... | None | None          | power off   | available        | False
|
+-----+-----+-----+-----+-----+-----+
+-----+

```

다음은 프로비저닝 프로세스에서 발생하는 몇 가지 일반적인 문제입니다.

- 결과 테이블에서 프로비저닝 상태 및 유지보수 열을 검토합니다. 다음을 확인하십시오.
 - 빈 테이블 또는 예상보다 적은 노드
 - 유지보수가 **True**로 설정되어 있는지 여부
 - 프로비저닝 상태가 **manageable**로 설정되어 있음. 이는 일반적으로 등록 또는 검색 프로세스에 문제가 있음을 나타냅니다. 예를 들어 유지보수가 자동으로 **True**로 설정되어 있으면 일반적으로 노드에서 잘못된 전원 관리 인증서를 사용하는 것입니다.
- 프로비저닝 상태가 **available**이면 베어 메탈 배포가 시작되기 전에 문제가 발생한 것입니다.
- 프로비저닝 상태가 **active**이고 전원 상태가 **power on**이면 베어 메탈 배포가 성공적으로 완료된 것입니다. 즉, 배포 후 구성 단계 중에 문제가 발생한 것입니다.
- 프로비저닝 상태가 노드에 대해 **wait call-back**이면 베어 메탈 프로비저닝 프로세스가 이 노드에 대해 아직 완료되지 않은 것입니다. 이 상태가 변경될 때까지 기다리고, 그렇지 않은 경우 실패한 노드의 가상 콘솔에 연결하고 출력을 확인합니다.
- 프로비저닝 상태가 **error** 또는 **deploy failed**이면 베어 메탈 프로비저닝이 이 노드에 대해 실패한 것입니다. 베어 메탈 노드의 세부 사항을 확인합니다.

```
$ ironic node-show [NODE UUID]
```

오류 설명이 포함된 **last_error** 필드를 찾습니다. 오류 메시지가 모호한 경우 로그를 사용하여 메시지를 명확히 파악할 수 있습니다.

```
$ sudo journalctl -u openstack-ironic-conductor -u openstack-ironic-api
```

- **wait timeout error**가 표시되고 노드 전원 상태가 **power on**이면 장애가 발생한 노드의 가상 콘솔에 연결하고 출력을 확인합니다.

11.4.3. 배포 후 구성

구성 단계 중에 여러 가지가 발생할 수 있습니다. 예를 들면 특정 **Puppet** 모듈이 설정 문제로 인해 완료되지 않을 수 있습니다. 이 섹션에서는 그러한 문제를 진단할 프로세스를 제공합니다.

오버클라우드 스택에서 리소스를 모두 나열하여 실패한 리소스를 확인합니다.

```
$ heat resource-list overcloud
```

이렇게 하면 모든 리소스 및 해당 상태 테이블이 표시됩니다. **CREATE_FAILED**가 있는 리소스를 찾습니다.

실패한 리소스를 표시합니다.

```
$ heat resource-show overcloud [FAILED RESOURCE]
```

resource_status_reason 필드에서 진단에 도움이 되는 정보를 확인합니다.

nova 명령을 사용하여 오버클라우드 노드의 IP 주소를 확인합니다.

```
$ nova list
```

배포된 노드 중 하나에 **heat-admin** 사용자로 로그인합니다. 예를 들어 스택의 리소스 목록에 **Controller** 노드에서 발생한 오류가 표시되면 **Controller** 노드에 로그인합니다. **heat-admin** 사용자에게 **sudo** 액세스 권한이 있습니다.

```
$ ssh heat-admin@192.168.24.14
```

os-collect-config 로그에서 가능한 오류 원인을 확인합니다.

```
$ sudo journalctl -u os-collect-config
```

경우에 따라 **nova**가 노드 전체를 배포하지 못할 수 있습니다. 오버클라우드 역할 유형 중 하나의 실패한 **OS::Heat::ResourceGroup**으로 이러한 상황을 알 수 있습니다. 이 경우 오류를 보려면 **nova**를 사용합니다.

```
$ nova list
$ nova show [SERVER ID]
```

표시되는 가장 일반적인 오류는 오류 메시지 **No valid host was found**입니다. 이 오류 문제 해결에 대한 자세한 내용은 [11.6절. “No Valid Host Found” 오류 문제 해결](#) 을 참조하십시오. 그 외의 경우는 문제 해결을 위해 다음 로그 파일을 확인하십시오.

- `/var/log/nova/*`
- `/var/log/heat/*`
- `/var/log/ironic/*`

시스템 하드웨어 및 구성에 대한 정보를 수집하는 **SOS** 툴셋을 사용합니다. 이 정보를 진단 목적 및 디버깅에 사용합니다. **SOS**는 일반적으로 기술자 및 개발자를 지원하는 데 사용됩니다. **SOS**는 언더클라우드와 오버클라우드 모두에서 유용합니다. **sos** 패키지를 설치합니다.

```
$ sudo yum install sos
```

보고서를 생성합니다.

```
$ sudo sosreport --all-logs
```

Controller 노드에 대한 배포 후 프로세스는 배포를 위해 5가지 기본 단계를 사용합니다.

표 11.1. Controller 노드 구성 단계

단계	설명
ControllerDeployment_Step1	Pacemaker, RabbitMQ, Memcached, Redis 및 Galera를 포함한 로드 밸런싱 소프트웨어의 초기 구성.
ControllerDeployment_Step2	Pacemaker 구성, HAProxy, MongoDB, Galera, Ceph 모니터 및 OpenStack Platform 서비스에 대한 데이터베이스 초기화를 포함한 초기 클러스터 구성.
ControllerDeployment_Step3	OpenStack Object Storage(swift)에 대한 초기 링 빌드. 모든 OpenStack Platform 서비스 (nova , neutron , cinder , sahara , ceilometer , heat , horizon , aodh , gnocchi) 구성.
ControllerDeployment_Step4	서비스 시작 순서 및 서비스 시작 매개 변수를 결정하는 제한 사항을 포함하여 Pacemaker에서 서비스 시작 설정 구성.
ControllerDeployment_Step5	OpenStack Identity(keystone)의 프로젝트, 역할 및 사용자의 초기 구성.

11.5. 프로비저닝 네트워크에서 IP 주소 충돌 문제 해결

대상 호스트가 이미 사용 중인 할당된 IP 주소이면 검색 및 배포 작업이 실패합니다. 이 문제가 발생하지 않도록 하려면 프로비저닝 네트워크의 포트 스캔을 수행하여 검색 IP 범위 및 호스트 IP 범위를 사용할 수 있는지 확인합니다.

언더클라우드 호스트에서 다음 단계를 수행합니다.

nmap을 설치합니다.

```
# yum install nmap
```

nmap을 사용하여 활성 주소에 대한 IP 주소 범위를 스캔합니다. 이 예에서는 192.168.24.0/24 범위를 스캔하고, 이 범위를 프로비저닝 네트워크의 IP 서브넷으로 교체합니다(CIDR 비트마스크 표기 사용).

```
# nmap -sn 192.168.24.0/24
```

nmap 스캔 출력을 검토합니다.

예를 들어 언더클라우드의 IP 주소 및 서브넷에 있는 다른 호스트가 표시됩니다. 활성 IP 주소가 undercloud.conf의 IP 범위와 충돌하는 경우 IP 주소 범위를 변경하거나 IP 주소가 사용 가능하도록 설정한 후에 오버클라우드 노드를 introspect하거나 배포해야 합니다.

```
# 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
```

11.6. "NO VALID HOST FOUND" 오류 문제 해결

/var/log/nova/nova-conductor.log에 다음 오류가 포함되는 경우가 있습니다.

```
NoValidHost: No valid host was found. There are not enough hosts
available.
```

이는 nova 스케줄러가 새 인스턴스 부팅에 적합한 베어 메탈 노드를 찾을 수 없음을 의미하며, 결국은 nova에서 찾아야 하는 리소스와 ironic에서 nova에 통지한 리소스가 일치하지 않음을 의미합니다. 이런 경우 다음을 확인하십시오.

1. introspection이 성공했는지 확인합니다. 그렇지 않으면 각 노드에 필수 ironic 노드 속성이 포함되어 있는지 확인합니다. 각 노드에 대해 다음을 수행합니다.

```
$ ironic node-show [NODE UUID]
```

properties JSON 필드에 cpus, cpu_arch, memory_mb 및 local_gb 키에 대한 올바른 값이 있는지 확인합니다.

2. 사용된 nova 플레이버가 필요한 노드 수에 대해 위의 ironic 노드 속성을 초과하지 않는지 확인합니다.

```
$ nova flavor-show [FLAVOR NAME]
```

3. **ironic node-list**에 따라 **available** 상태의 노드가 충분한지 확인합니다. **manageable** 상태의 노드란 일반적으로 실패한 **introspection**을 의미합니다.
4. 노드가 유지보수 모드에 있지 않은지 확인합니다. **ironic node-list**를 사용하여 확인합니다. 유지보수 모드로 자동으로 변경되는 노드는 전원 인증서가 잘못된 것입니다. 이러한 노드를 확인한 다음 유지보수 모드를 제거합니다.

```
$ ironic node-set-maintenance [NODE UUID] off
```

5. AHC(Automated Health Check) 툴을 사용하여 자동 노드 태깅을 수행하는 경우 각 플레이버/프로필에 해당하는 충분한 노드가 있는지 확인합니다. **properties** 필드에서 **ironic node-show**에 대한 **capabilities** 키를 확인합니다. 예를 들면 **Compute** 역할에 대해 태그된 노드에는 **profile:compute**가 포함되어 있어야 합니다.
6. **introspection** 후 노드 정보가 **ironic**에서 **nova**로 반영되는 데에는 시간이 걸립니다. **director** 툴에서 일반적으로 이를 수행합니다. 하지만 일부 단계를 수동으로 수행하는 경우 단시간 동안 노드를 **nova**에서 사용할 수 없습니다. 다음 명령을 사용하여 시스템에서 총 리소스를 확인합니다.

```
$ nova hypervisor-stats
```

11.7. 오버클라우드 생성 후 문제 해결

오버클라우드 생성 후 나중에 특정 오버클라우드 작업을 수행할 수 있습니다. 예를 들어 사용 가능한 노드를 확장하거나 문제가 발생한 노드를 교체할 수 있습니다. 이러한 작업을 수행할 때 특정 문제가 발생할 수 있습니다. 이 섹션에서는 생성 후 작업 실패 문제를 진단하고 해결하는 몇 가지 방법을 설명합니다.

11.7.1. 오버클라우드 스택 수정

director를 통해 **overcloud** 스택을 수정할 때 문제가 발생할 수 있습니다. 스택 수정의 예는 다음과 같습니다.

- 노드 확장
- 노드 제거
- 노드 교체

director가 요청한 노드 수의 가용성 확인, 추가 노드 프로비저닝 또는 기존 노드 제거 후 **Puppet** 구성 적용을 수행한다는 점에서 스택 수정은 스택 생성 프로세스와 비슷합니다. 다음은 **overcloud** 스택을 수정할 때 따라야 하는 몇 가지 지침입니다.

초기 단계로 **11.4.3절. “배포 후 구성”**에 설정된 지침을 따릅니다. 동일한 단계가 **overcloud heat** 스택 업데이트 시 문제를 진단하는 데에도 도움이 될 수 있습니다. 특히, 다음 명령을 사용하면 문제가 있는 리소스를 식별하는 데 도움이 됩니다.

heat stack-list --show-nested

모든 스택을 나열합니다. **--show-nested**는 모든 하위 스택과 해당 상위 스택을 표시합니다. 이 명령은 스택이 실패한 지점을 식별하는 데 도움이 됩니다.

heat resource-list overcloud

overcloud 스택의 모든 리소스 및 리소스의 현재 상태를 나열합니다. 이는 스택에서 오류를 일으키는 리소스를 식별하는 데 도움이 됩니다. **heat** 템플릿 컬렉션 및 **Puppet** 모듈에서 이 리소스 오류에 대해 해당 매개 변수와 구성으로 추적할 수 있습니다.

heat event-list overcloud

overcloud 스택과 관련된 모든 이벤트를 시간 순으로 나열합니다. 여기에는 스택에 있는 모든 리소스의 시작, 완료 및 오류가 포함됩니다. 이는 리소스 오류 지점을 식별하는 데 도움이 됩니다.

다음의 일부 섹션에서는 특정 노드 유형에 대한 문제 진단 지침을 제공합니다.

11.7.2. Controller 서비스 오류

오버클라우드 **Controller** 노드에는 대부분의 **Red Hat OpenStack Platform** 서비스가 포함되어 있습니다. 마찬가지로 고가용성 클러스터에서 여러 **Controller** 노드를 사용할 수 있습니다. 노드의 특정 서비스에 문제가 발생하면 고가용성 클러스터에서 특정 수준의 페일오버를 제공합니다. 하지만 문제가 발생한 서비스를 진단하여 오버클라우드가 최대 성능으로 작동하는지 확인해야 합니다.

Controller 노드는 **Pacemaker**를 사용하여 고가용성 클러스터에서 리소스 및 서비스를 관리합니다. **Pacemaker Configuration System(pcs)** 명령은 **Pacemaker** 클러스터를 관리하는 툴입니다. 클러스터의 **Controller** 노드에서 이 명령을 실행하여 구성 및 모니터링 기능을 수행합니다. 다음은 고가용성 클러스터에서 오버클라우드 서비스 문제를 해결하는 데 도움이 되는 몇 가지 명령입니다.

pcs status

활성화된 리소스, 실패한 리소스 및 온라인 노드를 포함하여 전체 클러스터에 대한 상태 개요를 제공합니다.

pcs resource show

리소스 목록 및 리소스의 해당 노드를 표시합니다.

pcs resource disable [resource]

특정 리소스를 중지합니다.

pcs resource enable [resource]

특정 리소스를 시작합니다.

pcs cluster standby [node]

노드를 대기 모드로 전환합니다. 노드를 클러스터에서 더 이상 사용할 수 없습니다. 클러스터에 영향을 주지 않고 특정 노드에서 유지보수를 수행하는 데 유용합니다.

pcs cluster unstandby [node]

대기 모드에서 노드를 제거합니다. 이 노드는 클러스터에서 다시 사용 가능하게 됩니다.

이러한 **Pacemaker** 명령을 사용하여 문제가 있는 구성 요소 및/또는 노드를 식별합니다. 구성 요소를 식별한 후에 **/var/log/**의 해당 구성 요소 로그 파일을 확인합니다.

11.7.3. Compute 서비스 오류

Compute 노드는 **Compute** 서비스를 사용하여 하이퍼바이저 기반 작업을 수행합니다. 이는 **Compute** 노드에 대한 기본 진단으로 이 서비스에 대한 문제를 해결할 수 있음을 의미합니다. 예를 들면 다음과 같습니다.

- 다음 **systemd** 기능을 사용하여 서비스 상태를 확인합니다.

```
$ sudo systemctl status openstack-nova-compute.service
```

마찬가지로 다음 명령을 사용하여 서비스에 대한 **systemd** 저널을 확인합니다.

```
$ sudo journalctl -u openstack-nova-compute.service
```

- **Compute** 노드에 대한 기본 로그 파일은 `/var/log/nova/nova-compute.log`입니다. **Compute** 노드 통신에 문제가 발생하는 경우 일반적으로 이 로그 파일을 사용하여 진단을 시작하는 것이 좋습니다.
- **Compute** 노드에서 유지보수를 수행하는 경우 기존 인스턴스를 호스트에서 운영 가능한 **Compute** 노드로 마이그레이션한 다음 해당 노드를 비활성화합니다. 노드 마이그레이션에 대한 자세한 내용은 [8.8절. “오버클라우드 Compute 노드에서 VM 마이그레이션”](#)을 참조하십시오.

11.7.4. Ceph Storage 서비스 오류

Red Hat Ceph Storage 클러스터에 발생한 문제에 대해서는 Red Hat Ceph Storage 구성 가이드의 [10장. 로깅 및 디버깅](#)을 참조하십시오. 이 섹션에서는 모든 **Ceph Storage** 서비스에 대한 진단 로그 정보를 제공합니다.

11.8. 언더클라우드 성능 조정

이 섹션의 지침은 언더클라우드의 성능 향상을 지원하기 위한 것입니다. 필요에 따라 권장사항을 구현하십시오.

- **Identity** 서비스(**keystone**)는 다른 **OpenStack** 서비스에 대한 액세스 제어에 토큰 기반 시스템을 사용합니다. 일정한 기간이 지나면 데이터베이스에는 사용하지 않는 많은 토큰이 누적됩니다. 기본 **cronjob**은 매일 토큰 테이블을 풀러시합니다. 환경을 모니터링하고 토큰 풀러시 간격을 필요에 따라 조정하는 것이 좋습니다. 언더클라우드에 대해 **crontab -u keystone -e**를 사용하여 간격을 조정할 수 있습니다. 이러한 변경은 일시적인 것으로 **openstack undercloud update**를 실행하면 이 **cronjob**이 다시 해당 기본값으로 재설정됩니다.
- **Heat**는 **openstack overcloud deploy**를 실행할 때마다 모든 템플릿 파일 복사본을 해당 데이터베이스의 **raw_template** 테이블에 저장합니다. **raw_template** 테이블은 이전의 모든 템플릿을 유지하고 크기를 늘립니다. **raw_templates** 테이블에서 사용하지 않는 템플릿을 제거하려면 사용하지 않고 하루 이상 데이터베이스에 있는 템플릿을 지우는 일별 **cronjob**을 생성하십시오.

```
0 04 * * * /bin/heat-manage purge_deleted -g days 1
```

- **openstack-heat-engine** 및 **openstack-heat-api** 서비스가 너무 많은 리소스를 사용하는 경우가 있습니다. 그런 경우 `/etc/heat/heat.conf`에서 **max_resources_per_stack=-1**을 설정하고 **heat** 서비스를 다시 시작하십시오.

```
$ sudo systemctl restart openstack-heat-engine openstack-heat-api
```

- **director**에 리소스가 부족하여 동시 모드 프로비저닝을 수행하지 못하는 경우가 있습니다. 기본값은 동시에 10개 노드입니다. 동시 노드 수를 줄이려면 `/etc/nova/nova.conf`에서 **max_concurrent_builds** 매개 변수를 10보다 작은 값으로 설정하고 **nova** 서비스를 다시 시작하십시오.

```
$ sudo systemctl restart openstack-nova-api openstack-nova-scheduler
```

- `/etc/my.cnf.d/server.cnf` 파일을 편집하십시오. 조정이 권장되는 값은 다음과 같습니다.

max_connections

데이터베이스에 대한 동시 연결 수입니다. 권장 값은 4096입니다.

innodb_additional_mem_pool_size

데이터베이스에서 데이터 사전 정보와 기타 내부 데이터 구조를 저장하는 데 사용하는 메모리 풀 크기(바이트)입니다. 언더클라우드에 대한 기본값은 일반적으로 8M이며, 이상적인 값은 20M입니다.

innodb_buffer_pool_size

데이터베이스에서 테이블 및 인덱스 데이터를 캐시하는 메모리 영역의 버퍼 풀 크기(바이트)입니다. 언더클라우드에 대한 기본값은 일반적으로 128M이며, 이상적인 값은 1000M입니다.

innodb_flush_log_at_trx_commit

커밋 관련 I/O 작업이 재정렬되고 배치로 수행되는 경우 가능한 고성능 및 커밋 작업을 위한 엄격한 ACID 준수 간 조정을 제어합니다. 1로 설정됩니다.

innodb_lock_wait_timeout

데이터베이스 트랜잭션이 중단될 때까지 행 잠금을 기다리는 시간(초)입니다. 50으로 설정됩니다.

innodb_max_purge_lag

이 변수는 제거 작업이 지연될 때 INSERT, UPDATE 및 DELETE 작업을 지연하는 방법을 제어합니다. 10000으로 설정합니다.

innodb_thread_concurrency

동시 작동하는 시스템 스레드 제한 수입니다. 각 CPU 및 디스크 리소스에 대해 적어도 두 개의 스레드를 제공하는 것이 좋습니다. 예를 들어 쿼드 코어 CPU 및 단일 디스크를 사용하는 경우 10개 스레드를 사용합니다.

- heat에 오버클라우드 생성을 수행할 충분한 작업자가 있는지 확인하십시오. 일반적으로 언더클라우드에 있는 CPU 수에 따라 다릅니다. 작업자 수를 수동으로 설정하려면 `/etc/heat/heat.conf` 파일을 편집하고, `num_engine_workers` 매개 변수를 필요한 작업자 수(4가 이상적임)로 설정한 다음 heat 엔진을 다시 시작합니다.

```
$ sudo systemctl restart openstack-heat-engine
```

11.9. 언더클라우드 및 오버클라우드 관련 중요 로그

문제를 해결할 때 다음 로그를 사용하여 언더클라우드 및 오버클라우드에 대한 정보를 찾습니다.

표 11.2. 언더클라우드 관련 중요 로그

정보	로그 위치
OpenStack Compute 로그	<code>/var/log/nova/nova-compute.log</code>
OpenStack Compute API 상호 작용	<code>/var/log/nova/nova-api.log</code>
OpenStack Compute Conductor 로그	<code>/var/log/nova/nova-conductor.log</code>
OpenStack Orchestration 로그	<code>heat-engine.log</code>
OpenStack Orchestration API 상호 작용	<code>heat-api.log</code>
OpenStack Orchestration CloudFormations 로그	<code>/var/log/heat/heat-api-cfn.log</code>
OpenStack Bare Metal Conductor 로그	<code>ironic-conductor.log</code>

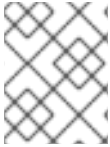
정보	로그 위치
OpenStack Bare Metal API 상호 작용	<code>ironic-api.log</code>
Introspection	<code>/var/log/ironic-inspector/ironic-inspector.log</code>
OpenStack Workflow Engine 로그	<code>/var/log/mistral/engine.log</code>
OpenStack Workflow Executor 로그	<code>/var/log/mistral/executor.log</code>
OpenStack Workflow API 상호 작용	<code>/var/log/mistral/api.log</code>

표 11.3. 오버클라우드 관련 중요 로그

정보	로그 위치
Cloud-Init 로그	<code>/var/log/cloud-init.log</code>
오버클라우드 구성(마지막 Puppet 실행 요약)	<code>/var/lib/puppet/state/last_run_summary.yaml</code>
오버클라우드 구성(마지막 Puppet 실행 보고서)	<code>/var/lib/puppet/state/last_run_report.yaml</code>
오버클라우드 구성(전체 Puppet 보고서)	<code>/var/lib/puppet/reports/overcloud-*/*</code>
오버클라우드 구성(각 Puppet 실행의 stdout)	<code>/var/run/heat-config/deployed/*-stdout.log</code>
오버클라우드 구성(각 Puppet 실행의 stderr)	<code>/var/run/heat-config/deployed/*-stderr.log</code>
고가용성 로그	<code>/var/log/pacemaker.log</code>

부록 A. SSL/TLS 인증서 구성

공용 엔드포인트에서 통신에 SSL/TLS를 사용하도록 언더클라우드를 구성할 수 있습니다. 하지만, 자체 인증 기관의 SSL 인증서를 사용하는 경우 인증서에 다음 섹션의 구성 단계가 필요합니다.



참고

오버클라우드 SSL/TLS 인증서 생성에 대해서는 *고급 오버클라우드 사용자 정의 가이드*에서 "[오버클라우드에서 SSL/TLS 활성화](#)"를 참조하십시오.

A.1. 서명 호스트 초기화

서명 호스트는 새 인증서를 생성하고 인증 기관을 통해 서명하는 호스트입니다. 선택한 서명 호스트에서 SSL 인증서를 생성한 적이 없는 경우 새 인증서에 서명할 수 있도록 호스트를 초기화해야 할 수 있습니다.

`/etc/pki/CA/index.txt` 파일은 서명한 모든 인증서 레코드를 저장합니다. 이 파일이 있는지 확인합니다. 없는 경우 빈 파일을 생성합니다.

```
$ sudo touch /etc/pki/CA/index.txt
```

`/etc/pki/CA/serial` 파일은 서명할 다음 인증서에 사용할 다음 일련 번호를 식별합니다. 이 파일이 있는지 확인합니다. 이 파일이 없는 경우 새 시작 값으로 새 파일을 생성합니다.

```
$ sudo echo '1000' | sudo tee /etc/pki/CA/serial
```

A.2. 인증 기관 생성

일반적으로는 외부 인증 기관을 통해 SSL/TLS 인증서에 서명합니다. 상황에 따라 고유한 인증 기관을 사용하려고 할 수 있습니다. 예를 들어 내부 전용 인증 기관을 사용하도록 설정할 수 있습니다.

예를 들어 인증 기관 역할을 할 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
```

`openssl req` 명령은 기관에 대한 특정 세부 사항을 요청합니다. 이러한 세부 사항을 입력합니다.

이렇게 하면 `ca.crt.pem`이라고 하는 인증 기관 파일이 생성됩니다.

A.3. 클라이언트에 인증 기관 추가

외부 클라이언트가 SSL/TLS를 사용하여 통신하려는 경우 Red Hat OpenStack Platform 환경에 액세스해야 하는 각 클라이언트에 인증 기관 파일을 복사합니다. 클라이언트에 복사했으면 클라이언트에서 다음 명령을 실행하여 인증 기관 신뢰 번들을 추가합니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

A.4. SSL/TLS 키 생성

언더클라우드 또는 오버클라우드 인증서를 생성할 여러 지점에서 사용하는 **SSL/TLS 키 (server.key.pem)**를 생성하려면 다음 명령을 실행합니다.

```
$ openssl genrsa -out server.key.pem 2048
```

A.5. SSL/TLS 인증서 서명 요청 생성

다음 절차에서는 언더클라우드 또는 오버클라우드에 대한 인증서 서명 요청을 생성합니다.

사용자 정의할 기본 **OpenSSL** 구성 파일을 복사합니다.

```
$ cp /etc/pki/tls/openssl.cnf .
```

사용자 정의 **openssl.cnf** 파일을 편집하고 **director**에 사용할 **SSL** 매개 변수를 설정합니다. 수정할 매개 변수 유형의 예는 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 중 하나로 설정합니다.

- IP 주소를 사용하여 **SSL/TLS**을 통해 액세스하는 경우 **undercloud_public_vip** 매개 변수를 **undercloud.conf**에 사용합니다.
- 정규화된 도메인 이름을 사용하여 **SSL/TLS**을 통해 액세스하는 경우 도메인 이름을 대신 사용합니다.

다음 항목을 포함하도록 **alt_names** 섹션을 편집합니다.

- **IP** - 클라이언트가 SSL을 통해 **director**에 액세스하는 IP 주소 목록입니다.
- **DNS** - 클라이언트가 SSL을 통해 **director**에 액세스하는 도메인 이름 목록입니다. 또한 공용 API IP 주소를 **alt_names** 섹션 끝에 DNS 항목으로 포함합니다.

openssl.cnf에 대한 자세한 내용을 보려면 **man openssl.cnf**를 실행합니다.

다음 명령을 실행하여 인증서 서명 요청을 생성합니다(**server.csr.pem**).

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out
server.csr.pem
```

-key 옵션에 대해 [A.4절. “SSL/TLS 키 생성”](#)에서 생성한 SSL/TLS 키를 반드시 포함하십시오.

다음 섹션에서 **server.csr.pem** 파일을 사용하여 SSL/TLS 인증서를 생성합니다.

A.6. SSL/TLS 인증서 생성

다음 명령은 언더클라우드 또는 오버클라우드에 대한 인증서를 생성합니다.

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in
server.csr.pem -out server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

이 명령에는 다음이 사용됩니다.

- **v3** 확장자를 지정하는 구성 파일. 이 파일을 **-config** 옵션으로 추가합니다.
- 인증서를 생성하고 인증 기관을 통해 서명하는 [A.5절. “SSL/TLS 인증서 서명 요청 생성”](#)의 인증서 서명 요청. 이 요청을 **-in** 옵션으로 추가합니다.
- [A.2절. “인증 기관 생성”](#)에서 생성한 인증 기관. 이 인증 기관에서 인증서에 서명하며, 이 인증 기관을 **-cert** 옵션으로 추가합니다.
- [A.2절. “인증 기관 생성”](#)에서 생성한 인증 기관 개인 키. 이 키를 **-keyfile** 옵션으로 추가합니다.

이렇게 하면 **server.crt.pem**이라는 인증서가 생성됩니다. 이 인증서를 [A.4절. “SSL/TLS 키 생성”](#)의 SSL/TLS 키와 함께 사용하여 SSL/TLS를 활성화합니다.

A.7. 언더클라우드에서 인증서 사용

다음 명령을 실행하여 인증서와 키를 함께 통합합니다.

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

이렇게 하면 **undercloud.pem** 파일이 생성됩니다. **undercloud.conf** 파일에서 **undercloud_service_certificate** 옵션에 대해 이 파일 위치를 지정합니다. 이 파일에는 HAProxy 틀에서 읽을 수 있도록 특수한 SELinux 컨텍스트도 필요합니다. 다음 예를 가이드로 사용하십시오.

```
$ sudo mkdir /etc/pki/instack-certs
$ sudo cp ~/undercloud.pem /etc/pki/instack-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/instack-certs(/.*)?"
$ sudo restorecon -R /etc/pki/instack-certs
```

undercloud.pem 파일 위치를 **undercloud.conf** 파일의 **undercloud_service_certificate** 옵션에 추가합니다. 예를 들면 다음과 같습니다.

```
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
```

또한 [A.2절. “인증 기관 생성”](#)의 인증 기관을 언더클라우드의 신뢰할 수 있는 인증 기관 목록에 추가하여 언더클라우드 내의 다른 서비스에서 인증 기관에 액세스할 수 있도록 합니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/  
$ sudo update-ca-trust extract
```

[4.6절. “Director 구성”](#)의 지침에 따라 언더클라우드를 계속 설치합니다.

부록 B. 전원 관리 드라이버

director는 기본적으로 전원 관리 컨트롤에 IPMI를 사용하지만 다른 전원 관리 유형도 지원합니다. 이 부록은 지원되는 전원 관리 기능 목록을 제공합니다. [5.1절. “오버클라우드에 노드 등록”](#)에 대해 이러한 전원 관리 설정을 사용하십시오.

B.1. DRAC(DELL REMOTE ACCESS CONTROLLER)

DRAC는 전원 관리 및 서버 모니터링을 포함하여 대역 외 원격 관리 기능을 제공하는 인터페이스입니다.

pm_type

이 옵션을 **pxe_drac**로 설정합니다.

pm_user; pm_password

DRAC 사용자 이름 및 암호입니다.

pm_addr

DRAC 호스트의 IP 주소입니다.

B.2. ILO(INTEGRATED LIGHTS-OUT)

Hewlett-Packard의 iLO는 전원 관리 및 서버 모니터링을 포함하여 대역 외 원격 관리 기능을 제공하는 인터페이스입니다.

pm_type

이 옵션을 **pxe_ilo**로 설정합니다.

pm_user; pm_password

iLO 사용자 이름 및 암호입니다.

pm_addr

iLO 인터페이스의 IP 주소입니다.

- **/etc/ironic/ironic.conf** 파일을 편집하고 **pxe_ilo**를 **enabled_drivers** 옵션에 추가하여 이 드라이버를 활성화합니다.
- **director**에는 iLO에 대한 추가 유틸리티 세트도 필요합니다. **python-proliantutils** 패키지를 설치하고 **openstack-ironic-conductor** 서비스를 다시 시작합니다.

```
$ sudo yum install python-proliantutils
$ sudo systemctl restart openstack-ironic-conductor.service
```

- 성공적인 **introspection**을 위해서는 HP 노드가 2015 펌웨어 버전이어야 합니다. **director**는 펌웨어 버전 1.85(2015년 5월 13일)를 사용하는 노드에서 성공적으로 테스트되었습니다.
- 공유 iLO 포트 사용은 지원되지 않습니다.

B.3. IBOOT

Dataprobe의 iBoot는 시스템에 원격 관리를 제공하는 전원 장치입니다.

pm_type

이 옵션을 **pxe_iboot**로 설정합니다.

pm_user; pm_password

iBoot 사용자 이름 및 암호입니다.

pm_addr

iBoot 인터페이스의 IP 주소입니다.

pm_relay_id(선택 사항)

호스트에 대한 iBoot 릴레이 ID입니다. 기본값은 1입니다.

pm_port(선택 사항)

iBoot 포트입니다. 기본값은 9100입니다.

- `/etc/ironic/ironic.conf` 파일을 편집하고 `pxe_iboot`를 `enabled_drivers` 옵션에 추가하여 이 드라이버를 활성화합니다.

B.4. CISCO UCS(UNIFIED COMPUTING SYSTEM)

Cisco의 UCS는 계산, 네트워크, 스토리지 액세스 및 가상화 리소스를 통합한 데이터 센터 플랫폼입니다. 이 드라이버는 UCS에 연결된 베어 메탈 시스템의 전원 관리에 중점을 둡니다.

pm_type

이 옵션을 `pxe_ucs`로 설정합니다.

pm_user; pm_password

UCS 사용자 이름 및 암호입니다.

pm_addr

UCS 인터페이스의 IP 주소입니다.

pm_service_profile

사용할 UCS 서비스 프로파일입니다. 일반적으로 `org-root/ls-[service_profile_name]` 형식을 사용합니다. 예를 들면 다음과 같습니다.

```
"pm_service_profile": "org-root/ls-Nova-1"
```

- `/etc/ironic/ironic.conf` 파일을 편집하고 `pxe_ucs`를 `enabled_drivers` 옵션에 추가하여 이 드라이버를 활성화합니다.
- `director`에는 UCS에 대한 추가 유틸리티 세트도 필요합니다. `python-UcsSdk` 패키지를 설치하고 `openstack-ironic-conductor` 서비스를 다시 시작합니다.

```
$ sudo yum install python-UcsSdk
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.5. FUJITSU IRMC(INTEGRATED REMOTE MANAGEMENT CONTROLLER)

Fujitsu의 iRMC는 통합된 LAN 연결 및 확장된 기능이 있는 BMC(Baseboard Management Controller)입니다. 이 드라이버는 iRMC에 연결된 베어 메탈 시스템의 전원 관리에 중점을 둡니다.



중요

iRMC S4 이상이 필요합니다.

pm_type

이 옵션을 **pxe_irmc**로 설정합니다.

pm_user; pm_password

iRMC 인터페이스에 대한 사용자 이름 및 암호입니다.

pm_addr

iRMC 인터페이스의 IP 주소입니다.

pm_port(선택 사항)

iRMC 작업에 사용할 포트입니다. 기본값은 **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 **scc**를 사용합니다. 기본값은 **ipmitool**입니다.

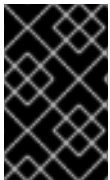
- **/etc/ironic/ironic.conf** 파일을 편집하고 **pxe_irmc**를 **enabled_drivers** 옵션에 추가하여 이 드라이버를 활성화합니다.
- SCCI를 센서 방법으로 활성화한 경우 **director**에는 추가 유틸리티 세트도 필요합니다. **python-scciclient** 패키지를 설치하고 **openstack-ironic-conductor** 서비스를 다시 시작합니다.

```
$ yum install python-scciclient
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.6. VBMC(VIRTUAL BARE METAL CONTROLLER)

director는 가상 머신을 KVM 호스트의 노드로 사용할 수 있습니다. 에뮬레이션된 IPMI 장치를 통해 해당 전원 관리를 제어합니다. 이를 통해 [5.1절. “오버클라우드에 노드 등록”](#)의 표준 IPMI 매개 변수를 사용할 수 있지만, 가상 노드에만 사용 가능합니다.

이 전원 관리 방법은 현재 더 이상 사용되지 않는 [B.7절. “SSH 및 Virsh”](#)의 방법을 대체합니다.



중요

이 옵션은 가상 머신을 베어 메탈 노드 대신 사용합니다. 즉, 테스트 및 평가 목적에만 사용할 수 있습니다. Red Hat OpenStack Platform 엔터프라이즈 환경에는 사용하지 않는 것이 좋습니다.

KVM 호스트 구성

KVM 호스트에서 OpenStack Platform 리포지토리를 활성화하고 **python-virtualbmc** 패키지를 설치합니다.

```
$ sudo subscription-manager repos --enable=rhel-7-server-openstack-11-rpms
$ sudo yum install -y python-virtualbmc
```

vbmc 명령을 사용하여 각 가상 머신에 대한 가상 BMC(베어 메탈 컨트롤러)를 생성합니다. 예를 들어 **Node01** 및 **Node02**라는 가상 머신에 대한 BMC를 생성하려는 경우 다음 명령을 실행합니다.

```
$ vbmc add Node01 --port 6230 --username admin --password p455w0rd!
$ vbmc add Node02 --port 6231 --username admin --password p455w0rd!
```

이는 각 BMC에 액세스할 포트를 정의하며 각 BMC의 인증 세부 사항을 설정합니다.



참고

각 가상 머신에 다른 포트를 사용합니다. 1025 미만의 포트 번호에는 시스템에서 **root** 권한이 있어야 합니다.

다음 명령을 사용하여 각 BMC를 시작합니다.

```
$ vbmc start Node01
$ vbmc start Node02
```



참고

KVM 호스트를 재부팅한 후 이 단계를 반복해야 합니다.

노드 등록

노드 등록 파일(/home/stack/instackenv.json)에 다음 매개 변수를 사용합니다.

pm_type

이 옵션을 **pxe_ipmitool**로 설정합니다.

pm_user; pm_password

노드의 가상 BMC 장치에 대한 IPMI 사용자 이름 및 암호입니다.

pm_addr

노드가 포함된 KVM 호스트의 IP 주소입니다.

pm_port

KVM 호스트에서 특정 노드에 액세스할 포트입니다.

mac

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

예:

```
{
  "nodes": [
    {
      "pm_type": "pxe_ipmitool",
      "mac": [
        "aa:aa:aa:aa:aa:aa"
      ],
      "pm_user": "admin",
      "pm_password": "p455w0rd!",
      "pm_addr": "192.168.0.1",
      "pm_port": "6230",
      "name": "Node01"
    },
  ],
}
```

```
{
  "pm_type": "pxe_ipmitool",
  "mac": [
    "bb:bb:bb:bb:bb:bb"
  ],
  "pm_user": "admin",
  "pm_password": "p455w0rd!",
  "pm_addr": "192.168.0.1",
  "pm_port": "6231",
  "name": "Node02"
}
```

기존 노드 마이그레이션

더 이상 사용되지 않는 **pxe_ssh** 드라이버를 사용하는 기존 노드를 새 가상 BMC 방법을 사용하도록 마이그레이션합니다. 다음 명령은 **pxe_ipmitool** 드라이버와 해당 매개 변수를 사용하도록 노드를 설정하는 예입니다.

```
openstack baremetal node set Node01 \
  --driver pxe_ipmitool \
  --driver-info ipmi_address=192.168.0.1 \
  --driver-info ipmi_port=6230 \
  --driver-info ipmi_username="admin" \
  --driver-info ipmi_password="p455w0rd!"
```

B.7. SSH 및 VIRSH

director는 SSH를 통해 **libvirt**를 실행하는 KVM 호스트에 액세스하고 가상 머신을 노드로 사용할 수 있습니다. **director**는 **virsh**를 사용하여 이러한 노드의 전원 관리를 제어합니다.



중요

이제 **B.6절. “VBMC(Virtual Bare Metal Controller)”**에 나온 방법을 사용하므로 이 옵션은 더 이상 사용되지 않습니다. 사용되지 않는 이 드라이버를 계속 사용하려는 경우 테스트 및 평가 목적으로만 사용할 수 있습니다. **Red Hat OpenStack Platform** 엔터프라이즈 환경에는 사용하지 않는 것이 좋습니다.

pm_type

이 옵션을 **pxe_ssh**로 설정합니다.

pm_user; pm_password

SSH 개인 키의 SSH 사용자 이름 및 콘텐츠입니다. CLI 툴을 사용하여 노드를 등록하는 경우 개인 키는 이스케이프 문자(**\n**)로 새 줄이 교체된 줄에 있어야 합니다. 예를 들면 다음과 같습니다.

```
-----BEGIN RSA PRIVATE KEY-----\nMIIEEogIBAAKCAQEA .... kk+WXt9Y=\n-----
END RSA PRIVATE KEY-----
```

SSH 공용 키를 **libvirt** 서버의 **authorized_keys** 컬렉션에 추가합니다.

pm_addr

가상 호스트의 IP 주소입니다.

- **libvirt**를 호스팅하는 서버에는 **pm_password** 속성으로 설정된 공용 키와 **SSH** 키 쌍이 필요합니다.
- 선택한 **pm_user**가 **libvirt** 환경에 완전히 액세스할 수 있는지 확인합니다.

B.8. 페이크 PXE 드라이버

이 드라이버를 사용하면 전원 관리 없이 베어 메탈 장치를 사용할 수 있습니다. 따라서 **director**가 등록된 베어 메탈 장치를 제어하지 않으므로 **introspect** 및 배포 프로세스의 특정 시점에서 전원의 수동 제어가 필요합니다.



중요

이 옵션은 테스트 및 평가 목적으로만 사용할 수 있습니다. Red Hat OpenStack Platform 엔터프라이즈 환경에는 사용하지 않는 것이 좋습니다.

pm_type

이 옵션을 **fake_pxe**로 설정합니다.

- 이 드라이버는 전원 관리를 제어하지 않으므로 인증 세부 사항을 사용하지 않습니다.
- **/etc/ironic/ironic.conf** 파일을 편집하고 **fake_pxe**를 **enabled_drivers** 옵션에 추가하여 이 드라이버를 활성화합니다. 파일을 편집한 후에 베어 메탈 서비스를 다시 시작합니다.

```
$ sudo systemctl restart openstack-ironic-api openstack-ironic-conductor
```

- 노드에서 **introspection**을 수행할 때 **openstack baremetal introspection bulk start** 명령을 실행한 후 노드의 전원을 수동으로 켭니다.
- 오버클라우드 배포를 수행할 때 **ironic node-list** 명령을 사용하여 노드 상태를 확인합니다. 노드 상태가 **deploying**에서 **deploy wait-callback**으로 변경될 때까지 기다린 다음 노드의 전원을 수동으로 켭니다.
- 오버클라우드 프로비저닝 프로세스가 완료되면 노드를 다시 시작합니다. 프로비저닝이 완료되었는지 확인하려면 **ironic node-list** 명령을 사용하여 노드 상태를 확인하고, 노드 상태가 **active**로 변경될 때까지 기다린 다음 모든 오버클라우드 노드를 수동으로 재부팅합니다.

부록 C. 전체 디스크 이미지

기본 오버클라우드 이미지는 플랫 파티션 이미지입니다. 즉, 이미지 자체에 파티셔닝 정보 또는 부트로더가 포함되어 있지 않습니다. **director**는 부팅할 때 별도의 커널 및 램디스크를 사용하고, 오버클라우드 이미지를 디스크에 쓸 때 기본 파티셔닝 레이아웃을 생성합니다. 하지만 파티셔닝 레이아웃 및 부트로더를 포함한 전체 디스크 이미지를 생성할 수 있습니다.

C.1. 전체 디스크 이미지 생성

overcloud-full.qcow2 플랫 파티션 이미지에서 전체 디스크 이미지를 생성하는 작업에는 다음 단계가 포함됩니다.

1. 전체 디스크 이미지를 위해 **overcloud-full** 플랫 파티션을 기본 파티션으로 엽니다.
2. 새 전체 디스크 이미지를 원하는 크기로 생성합니다. 이 예에서는 **10GB** 이미지를 사용합니다.
3. 전체 디스크 이미지에서 파티션 및 볼륨을 생성합니다. 원하는 전체 디스크 이미지에 필요한 만큼 파티션과 볼륨을 생성합니다. 이 예에서는 **boot**에 대해 격리된 파티션을 생성하고 파일 시스템의 다른 콘텐츠에 대해 논리적 볼륨을 생성합니다.
4. 파티션 및 볼륨에 초기 파일 시스템을 생성합니다.
5. 플랫 파티션 파일 시스템을 마운트하고 콘텐츠를 전체 디스크 이미지의 올바른 파티션에 복사합니다.
6. **fstab** 콘텐츠를 생성하여 전체 디스크 이미지의 **/etc/fstab**에 저장합니다.
7. 모든 파일 시스템을 마운트 해제합니다.
8. 전체 디스크 이미지에서만 파티션을 마운트합니다. **/**에서 마운트된 **root** 파티션으로 시작하고 해당 디렉터리에 다른 파티션을 마운트합니다.
9. 셸 명령을 사용해 부트로더를 설치하여 **grub2-install** 및 **grub2-mkconfig**를 전체 디스크 이미지에서 실행합니다. 이렇게 하면 **grub2** 부트로더가 전체 디스크 이미지에 설치됩니다.
10. **dracut**를 업데이트하여 논리 볼륨 관리 지원을 추가합니다.
11. 모든 파일 시스템을 마운트 해제하고 이미지를 종료합니다.

C.2. 전체 디스크 이미지를 수동으로 생성

이미지 생성에 권장되는 툴은 다음 명령을 사용하여 설치하는 **guestfish**입니다.

```
$ sudo yum install -y guestfish
```

설치되면 **guestfish** 대화형 셸을 실행합니다.

```
$ guestfish
```

```
Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.
```

```
Type: 'help' for help on commands
      'man' to read the manual
```

```
'quit' to quit the shell
```

```
><fs>
```

guestfish 사용에 대한 자세한 내용은 Red Hat Enterprise Linux 7에 대한 *가상화 배포 및 관리 가이드*의 "[Guestfish 쉘](#)"을 참조하십시오.

C.3. 자동으로 전체 디스크 이미지 생성

다음 Python 스크립트는 **guestfish** 라이브러리를 사용하여 전체 디스크 이미지를 자동으로 생성합니다.

```
#!/usr/bin/env python
import guestfs
import os

# remove old generated drive
try:
    os.unlink("/home/stack/images/overcloud-full-partitioned.qcow2")
except:
    pass

g = guestfs.GuestFS(python_return_dict=True)

# import old and new images
print("Creating new repartitioned image")
g.add_drive_opts("/home/stack/images/overcloud-full.qcow2",
format="qcow2", readonly=1)
g.disk_create("/home/stack/images/overcloud-full-partitioned.qcow2",
"qcow2", 10.2 * 1024 * 1024 * 1024) #10.2G
g.add_drive_opts("/home/stack/images/overcloud-full-partitioned.qcow2",
format="qcow2", readonly=0)
g.launch()

# create the partitions for new image
print("Creating the initial partitions")
g.part_init("/dev/sdb", "mbr")
g.part_add("/dev/sdb", "primary", 2048, 616448)
g.part_add("/dev/sdb", "primary", 616449, -1)

g.pvcreate("/dev/sdb2")
g.vgcreate("vg", ['/dev/sdb2', ])
g.lvcreate("var", "vg", 5 * 1024)
g.lvcreate("tmp", "vg", 500)
g.lvcreate("swap", "vg", 250)
g.lvcreate("home", "vg", 100)
g.lvcreate("root", "vg", 4 * 1024)
g.part_set_bootable("/dev/sdb", 1, True)

# add filesystems to volumes
print("Adding filesystems")
ids = {}
keys = [ 'var', 'tmp', 'swap', 'home', 'root' ]
volumes = ['/dev/vg/var', '/dev/vg/tmp', '/dev/vg/swap', '/dev/vg/home',
'/dev/vg/root']
```

```

swap_volume = volumes[2]

count = 0
for volume in volumes:
    if count!=2:
        g.mkfs('ext4', volume)
        ids[keys[count]] = g.vfs_uuid(volume)
    count +=1

# create filesystem on boot and swap
g.mkfs('ext4', '/dev/sdb1')
g.mkswap_opts(volumes[2])
ids['swap'] = g.vfs_uuid(volumes[2])

# mount drives and copy content
print("Start copying content")
g.mkmountpoint('/old')
g.mkmountpoint('/root')
g.mkmountpoint('/boot')
g.mkmountpoint('/home')
g.mkmountpoint('/var')
g.mount('/dev/sda', '/old')

g.mount('/dev/sdb1', '/boot')
g.mount(volumes[4], '/root')
g.mount(volumes[3], '/home')
g.mount(volumes[0], '/var')

# copy content to root
results = g.ls('/old/')
for result in results:
    if result not in ('boot', 'home', 'tmp', 'var'):
        print("Copying %s to root" % result)
        g.cp_a('/old/%s' % result, '/root/')

# copy extra content
folders_to_copy = ['boot', 'home', 'var']
for folder in folders_to_copy:
    results = g.ls('/old/%s/' % folder)
    for result in results:
        print("Copying %s to %s" % (result, folder))
        g.cp_a('/old/%s/%s' % (folder, result),
                '/%s/' % folder)

# create /etc/fstab file
print("Generating fstab content")
fstab_content = """
UUID={boot_id} /boot ext4 defaults 0 2
UUID={root_id} / ext4 defaults 0 1
UUID={swap_id} none swap sw 0 0
UUID={tmp_id} /tmp ext4 defaults 0 2
UUID={home_id} /home ext4 defaults 0 2
UUID={var_id} /var ext4 defaults 0 2
""".format(
    boot_id=g.vfs_uuid('/dev/sdb1'),
    root_id=ids['root'],

```

```

    swap_id=ids['swap'],
    tmp_id=ids['tmp'],
    home_id=ids['home'],
    var_id=ids['var'])

g.write('/root/etc/fstab', fstab_content)

# unmount filesystems
g.umount('/root')
g.umount('/boot')
g.umount('/old')
g.umount('/var')

# mount in the right directories to install bootloader
print("Installing bootloader")
g.mount(volumes[4], '/')
g.mkdir('/boot')
g.mkdir('/var')
g.mount('/dev/sdb1', '/boot')
g.mount(volumes[0], '/var')

# do a selinux relabel
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts',
'/', force=True)
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts',
'/var', force=True)

g.sh('grub2-install --target=i386-pc /dev/sdb')
g.sh('grub2-mkconfig -o /boot/grub2/grub.cfg')

# create dracut.conf file
dracut_content = ""
add_dracutmodules+="lvm crypt"
""
g.write('/etc/dracut.conf', dracut_content)

# update initramfs to include lvm and crypt
kernels = g.ls('/lib/modules')
for kernel in kernels:
    print("Updating dracut to include modules in kernel %s" % kernel)
    g.sh('dracut -f /boot/initramfs-%s.img %s --force' % (kernel, kernel))
g.umount('/boot')
g.umount('/var')
g.umount('/')

# close images
print("Finishing image")
g.shutdown()
g.close()

```

이 스크립트를 언더클라우드에 실행 파일로 저장하고 **stack** 사용자로 실행합니다.

```
$ ./whole-disk-image.py
```

이렇게 하면 플랫폼 파티션 이미지에서 전체 디스크 이미지가 자동으로 생성됩니다. 전체 디스크 이미지 생성이 완료되면 이전 **overcloud-full.qcow2** 이미지를 교체합니다.


```
$ mv ~/images/overcloud-full.qcow2 ~/images/overcloud-full-old.qcow2
$ cp ~/images/overcloud-full-partitioned.qcow2 ~/images/overcloud-
full.qcow2
```

이제 전체 디스크 이미지를 다른 이미지와 함께 업로드할 수 있습니다.

C.4. 전체 디스크 이미지의 볼륨 암호화

guestfish를 사용하여 전체 디스크 이미지의 볼륨을 암호화할 수도 있습니다. 이렇게 하면 **luks-format** 하위 명령을 사용해야 합니다. 이 명령은 현재 볼륨을 지우고 암호화된 볼륨을 생성합니다.

다음 Python 스크립트는 C.3절. “자동으로 전체 디스크 이미지 생성”에 있는 스크립트의 수정된 버전입니다. 이 새 스크립트는 **home** 볼륨을 암호화합니다.

```
#!/usr/bin/env python
import binascii
import guestfs
import os

# remove old generated drive
try:
    os.unlink("/tmp/overcloud-full-partitioned.qcow2")
except:
    pass

g = guestfs.GuestFS(python_return_dict=True)

# import old and new images
print("Creating new repartitioned image")
g.add_drive_opts("/tmp/overcloud-full.qcow2", format="qcow2", readonly=1)
g.disk_create("/tmp/overcloud-full-partitioned.qcow2", "qcow2", 10 * 1024
* 1024 * 1024) #10G
g.add_drive_opts("/tmp/overcloud-full-partitioned.qcow2", format="qcow2",
readonly=0)
g.launch()

# create the partitions for new image
print("Creating the initial partitions")
g.part_init("/dev/sdb", "mbr")
g.part_add("/dev/sdb", "primary", 2048, 616448)
g.part_add("/dev/sdb", "primary", 616449, -1)

g.pvcreate("/dev/sdb2")
g.vgcreate("vg", ['/dev/sdb2', ])
g.lvcreate("var", "vg", 4400)
g.lvcreate("tmp", "vg", 500)
g.lvcreate("swap", "vg", 250)
g.lvcreate("home", "vg", 100)
g.lvcreate("root", "vg", 4000)
g.part_set_bootable("/dev/sdb", 1, True)

# encrypt home partition and write keys
print("Encrypting volume")
random_content = binascii.b2a_hex(os.urandom(1024))
g.luks_format('/dev/vg/home', random_content, 0)
```

```

# open the encrypted volume
volumes = ['/dev/vg/var', '/dev/vg/tmp', '/dev/vg/swap',
            '/dev/mapper/encryptedhome', '/dev/vg/root']

g.luks_open('/dev/vg/home', random_content, 'encryptedhome')
g.vgscan()
g.vg_activate_all(True)

# add filesystems to volumes
print("Adding filesystems")
ids = {}
keys = [ 'var', 'tmp', 'swap', 'home', 'root' ]
swap_volume = volumes[2]

count = 0
for volume in volumes:
    if count!=2:
        g.mkfs('ext4', volume)
        if keys[count] == 'home':
            ids['home'] = g.vfs_uuid('/dev/vg/home')
        else:
            ids[keys[count]] = g.vfs_uuid(volume)
        count +=1

# create filesystem on boot and swap
g.mkfs('ext4', '/dev/sdb1')
g.mkswap_opts(volumes[2])
ids['swap'] = g.vfs_uuid(volumes[2])

# mount drives and copy content
print("Start copying content")
g.mkmountpoint('/old')
g.mkmountpoint('/root')
g.mkmountpoint('/boot')
g.mkmountpoint('/home')
g.mkmountpoint('/var')
g.mount('/dev/sda', '/old')

g.mount('/dev/sdb1', '/boot')
g.mount(volumes[4], '/root')
g.mount(volumes[3], '/home')
g.mount(volumes[0], '/var')

# copy content to root
results = g.ls('/old/')
for result in results:
    if result not in ('boot', 'home', 'tmp', 'var'):
        print("Copying %s to root" % result)
        g.cp_a('/old/%s' % result, '/root/')

# copy extra content
folders_to_copy = ['boot', 'home', 'var']
for folder in folders_to_copy:
    results = g.ls('/old/%s/' % folder)
    for result in results:

```

```

    print("Copying %s to %s" % (result, folder))
    g.cp_a('/old/%s/%s' % (folder, result),
          '/%s/' % folder)

# write keyfile for encrypted volume
g.write('/root/root/home_keyfile', random_content)
g.chmod(0400, '/root/root/home_keyfile')

# generate mapper for encrypted home
mapper = """
home UUID={home_id} /root/home_keyfile
""".format(home_id=ids['home'])
g.write('/root/etc/crypttab', mapper)

# create /etc/fstab file
print("Generating fstab content")
fstab_content = """
UUID={boot_id} /boot ext4 defaults 1 2
UUID={root_id} / ext4 defaults 1 1
UUID={swap_id} none swap sw 0 0
UUID={tmp_id} /tmp ext4 defaults 1 2
UUID={var_id} /var ext4 defaults 1 2
/dev/mapper/home /home ext4 defaults 1 2
""".format(
    boot_id=g.vfs_uuid('/dev/sdb1'),
    root_id=ids['root'],
    swap_id=ids['swap'],
    tmp_id=ids['tmp'],
    home_id=ids['home'],
    var_id=ids['var'])

g.write('/root/etc/fstab', fstab_content)

# umount filesystems
g.umount('/root')
g.umount('/boot')
g.umount('/old')
g.umount('/var')
g.umount('/home')

# close encrypted volume
g.luks_close('/dev/mapper/cryptedhome')

# mount in the right directories to install bootloader
print("Installing bootloader")
g.mount(volumes[4], '/')
g.mkdir('/boot')
g.mkdir('/var')
g.mount('/dev/sdb1', '/boot')
g.mount(volumes[0], '/var')

# add rd.auto=1 on grub parameters
g.sh('sed -i "s/*.GRUB_CMDLINE_LINUX.*/GRUB_CMDLINE_LINUX=\\\"console=tty0
crashkernel=auto rd.auto=1\\\"/" /etc/default/grub')

g.sh('grub2-install --target=i386-pc /dev/sdb')

```

```

g.sh('grub2-mkconfig -o /boot/grub2/grub.cfg')

# create dracut.conf file
dracut_content = ""
add_dracutmodules+="lvm crypt"
""
g.write('/etc/dracut.conf', dracut_content)

# update initramfs to include lvm and crypt
kernels = g.ls('/lib/modules')
for kernel in kernels:
    print("Updating dracut to include modules in kernel %s" % kernel)
    g.sh('dracut -f /boot/initramfs-%s.img %s --force' % (kernel, kernel))

# do a selinux relabel
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts',
'/', force=True)
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts',
'/var', force=True)

g.umount('/boot')
g.umount('/var')
g.umount('/')

# close images
print("Finishing image")
g.shutdown()
g.close()

```

이 스크립트는 다음을 수행합니다.

- 키 생성(**random_content**)
- **/root/home_keyfile**에 암호화 키 저장
- **/root/home_keyfile**을 사용하여 자동으로 볼륨 암호를 해독할 **crypttab** 파일 생성

이 스크립트를 예로 사용하여 전체 디스크 이미지 생성 프로세스의 일부로 암호화된 볼륨을 생성하십시오.

C.5. 전체 디스크 이미지 업로드

전체 디스크 이미지를 업로드하려면 이미지 업로드 명령과 함께 **--whole-disk-image** 옵션을 사용합니다. 예를 들면 다음과 같습니다.

```

$ openstack overcloud image upload --whole-disk --image-path
/home/stack/images

```

이 명령은 **/home/stack/images**에서 이미지를 업로드하지만, **overcloud-full.qcow2** 파일을 전체 디스크 이미지로 처리합니다. 즉, 원하는 전체 디스크 이미지 이름을 **overcloud-full.qcow2**로 바꾼 후에 이미지 업로드 명령을 실행해야 합니다.

부록 D. 대체 부팅 모드

노드에 대한 기본 부팅 모드는 iPXE를 통한 BIOS입니다. 다음 섹션에는 **director**에서 노드를 프로비저닝하고 검사할 때 사용할 몇 가지 대체 부팅 모드가 요약되어 있습니다.

D.1. 표준 PXE

iPXE 부팅 프로세스는 HTTP를 사용하여 **introspection** 및 배포 이미지를 부팅합니다. 이전 시스템은 TFTP를 통해 부팅하는 표준 PXE 부팅만 지원할 수 있습니다.

iPXE에서 PXE로 변경하려면 **director** 호스트에서 **undercloud.conf** 파일을 편집하고 **ipxe_enabled**를 **False**로 설정합니다.

```
ipxe_enabled = False
```

이 파일을 저장하고 언더클라우드 설치를 실행합니다.

```
$ openstack undercloud install
```

이 프로세스에 대한 자세한 내용은 "[Red Hat OpenStack Platform director에서 PXE로 iPXE 변경](#)" 문서를 참조하십시오.

D.2. UEFI 부팅 모드

기본 부팅 모드는 기존 BIOS 모드입니다. 최신 시스템에 기존 BIOS 모드 대신 UEFI 부팅 모드가 필요할 수 있습니다. 이 경우 **undercloud.conf** 파일에서 다음을 설정합니다.

```
ipxe_enabled = True
inspection_enable_uefi = True
```

이 파일을 저장하고 언더클라우드 설치를 실행합니다.

```
$ openstack undercloud install
```

부팅 모드를 각 등록된 모드에 대한 **uefi**로 설정합니다. 예를 들어 기존 **boot_mode** 매개 변수를 **capabilities** 속성에 추가하거나 교체하려면 다음을 수행합니다.

```
$ NODE=<NODE NAME OR ID> ; openstack baremetal node set --property
capabilities="boot_mode:uefi,$(openstack baremetal node show $NODE -f json
-c properties | jq -r .properties.capabilities | sed "s/boot_mode:
[^\,]*,//g")" $NODE
```



참고

이 명령을 사용하여 **profile** 및 **boot_option** 기능이 유지되고 있는지 확인합니다.

또한 부팅 모드를 각 플레이버에 대한 **uefi**로 설정합니다. 예를 들면 다음과 같습니다.

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

부록 E. 자동 프로필 태깅

introspection 프로세스는 일련의 벤치마크 테스트를 수행합니다. **director**는 이러한 테스트의 데이터를 저장합니다. 여러 방법으로 이 데이터를 사용하는 정책 세트를 만들 수 있습니다. 예를 들면 다음과 같습니다.

- 정책을 통해 성능이 떨어지거나 안정적이지 않은 노드를 오버클라우드에서 사용하지 않도록 식별하고 격리할 수 있습니다.
- 정책을 통해 노드를 특정 프로필에 자동으로 태깅할 것인지 여부를 정의할 수 있습니다.

E.1. 정책 파일 구문

정책 파일은 규칙 세트가 포함된 **JSON** 포맷을 사용합니다. 각 규칙은 *description*, *condition* 및 *action*을 정의합니다.

설명

이는 일반 텍스트로된 규칙 설명입니다.

예:

```
"description": "A new rule for my node tagging policy"
```

조건

조건은 다음 키-값 패턴을 사용하여 평가를 정의합니다.

field

평가할 필드를 정의합니다. 필드 유형에 대해서는 [E.4절. “자동 프로필 태깅 속성”](#)을 참조하십시오.

op

평가에 사용할 작업을 정의합니다. 여기에 포함되는 작업은 다음과 같습니다.

- **eq** - 같음
- **ne** - 같지 않음
- **lt** - 보다 작음
- **gt** - 보다 큼
- **le** - 작거나 같음
- **ge** - 크거나 같음
- **in-net** - IP 주소가 지정된 네트워크에 있는지 확인
- **matches** - 지정된 정규 표현과 완전히 일치해야 함
- **contains** - 지정된 정규 표현식을 포함하는 값이 필요함
- **is-empty** - 빌드가 비어 있는지 확인

invert

평가 결과를 반전할지 여부를 정의하는 부울 값입니다.

multiple

여러 결과가 있는 경우 사용할 평가를 정의합니다. 예를 들면 다음과 같습니다.

- **any** - 임의 결과가 일치해야 함
- **all** - 모든 결과가 일치해야 함
- **first** - 첫 번째 결과가 일치해야 함

value

평가에 값을 정의합니다. 필드 및 작업 결과가 값에 있으면 조건이 **true** 결과를 반환합니다. 그렇지 않으면 조건이 **false**를 반환합니다.

예:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

동작

조건이 **true**로 반환되는 경우 작업이 수행됩니다. **action** 값에 따라 **action** 키와 추가 키를 사용합니다.

- **fail** - introspection이 실패합니다. 실패 메시지에 대한 **message** 매개 변수가 필요합니다.
- **set-attribute** - Ironic 노드에 대한 속성을 설정합니다. Ironic 속성에 대한 경로인 **path** 필드와(예: **/driver_info/ipmi_address**) 설정할 **value**가 필요합니다.
- **set-capability** - Ironic 노드에 대한 기능을 설정합니다. 각각 새 기능에 대한 이름 및 값인 **name**과 **value** 필드가 필요합니다. 이 동일한 기능에 대한 기존 값은 교체됩니다. 예를 들면 노드 프로필을 정의하는 데 이 값을 사용합니다.
- **extend-attribute** - **set-attribute**와 동일하지만, 기존 값을 목록으로 처리하고 값을 여기에 추가합니다. 옵션 **unique** 매개 변수가 **True**로 설정된 경우 지정된 값이 이미 목록에 있으면 아무것도 추가되지 않습니다.

예:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

E.2. 정책 파일 예

다음은 introspection 규칙이 적용되는 JSON 파일(**rules.json**) 예입니다.

```
[
  {
    "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 예는 다음 세 가지 규칙으로 구성되어 있습니다.

- 메모리가 4096MiB 이하인 경우 **introspection**이 실패합니다. 그러한 규칙을 적용하여 클라우드의 일부가 되지 않아야 하는 노드를 제외할 수 있습니다.
- 하드 드라이브 크기가 1TiB 이상인 노드가 무조건 할당된 **swift-storage** 프로파일입니다.
- 하드 드라이브 크기가 40GiB보다 크면서 1TiB 미만인 노드가 **Compute** 또는 **Controller** 노드가 될 수 있습니다. **openstack overcloud profiles match** 명령으로 나중에 최종 선택을 할 수 있도록 두 가지 기능(**compute_profile** 및 **control_profile**)을 할당합니다. 그렇게 하려면 기존 프로파일 기능을 제거하고, 그렇게 하지 않는 경우 기존 프로파일 기능이 우선 순위를 가집니다.

다른 노드는 변경되지 않습니다.



참고

introspection 규칙을 사용하여 **profile** 기능을 할당하면 기존 값을 항상 덮어씁니다. 하지만 기존 프로파일 기능을 사용하는 노드의 경우 **[PROFILE]_profile** 기능이 무시됩니다.

E.3. 정책 파일 가져오기

다음 명령을 사용하여 정책 파일을 **director**로 가져옵니다.

```
$ openstack baremetal introspection rule import rules.json
```

그런 다음 **introspection** 프로세스를 실행합니다.

```
$ openstack baremetal introspection bulk start
```

introspection이 완료되면 노드 및 할당된 해당 프로파일을 확인합니다.

```
$ openstack overcloud profiles list
```

introspection 규칙에 실수가 있는 경우 모두 삭제할 수 있습니다.

```
$ openstack baremetal introspection rule purge
```

E.4. 자동 프로파일 태깅 속성

자동 프로파일 태깅은 각 조건의 **field** 속성에 대해 다음 노드 속성을 평가합니다.

속성	설명
memory_mb	노드의 메모리 크기(MB)입니다.
cpus	노드의 CPU에 대한 총 코어 수입니다.
cpu_arch	노드 CPU의 아키텍처입니다.
local_gb	노드 root 디스크의 총 스토리지 공간입니다. 노드에 대한 root 디스크 설정에 대해서는 5.4절. “노드의 Root 디스크 정의” 를 참조하십시오.

부록 F. 보안 강화

다음 섹션에서는 언더클라우드의 보안 강화를 위한 몇 가지 제안 사항을 제공합니다.

F.1. HAPROXY에 대한 SSL/TLS 암호화 방식 및 규칙 변경

언더클라우드에서 SSL/TLS를 활성화한 경우(4.6절. “Director 구성” 참조) HAProxy 구성에 사용된 SSL/TLS 암호화 방식 및 규칙을 강화할 수 있습니다. 그러면 **POODLE 취약성**과 같은 SSL/TLS 취약성을 방지하는 데 도움이 됩니다.

hieradata_override 언더클라우드 구성 옵션을 사용하여 다음 hieradata를 설정합니다.

tripleo::haproxy::ssl_cipher_suite

HAProxy에서 사용할 암호화 방식 세트입니다.

tripleo::haproxy::ssl_options

HAProxy에서 사용할 SSL/TLS 규칙입니다.

예를 들면 다음 암호화 방식 및 규칙을 사용할 수 있습니다.

- 암호화 방식: **ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-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**
- 규칙: **no-sslsv3 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-sslsv3 no-tls-tickets
```



참고

암호화 방식 컬렉션은 연속된 한 줄로 되어 있습니다.

openstack undercloud install을 실행하기 전에 생성한 **hieradata** 덮어쓰기 파일을 사용하도록 **undercloud.conf** 파일에서 **hieradata_override** 매개 변수를 설정합니다.

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```