Red Hat OpenStack Platform 13

고급 오버클라우드 사용자 정의

Red Hat OpenStack Platform director를 사용하여 고급 기능을 구성하는 방법

Last Updated: 2023-05-25

Red Hat OpenStack Platform 13 고급 오버클라우드 사용자 정의

Red Hat OpenStack Platform director를 사용하여 고급 기능을 구성하는 방법

OpenStack Team
rhos-docs@redhat.com

법적 공지

초록

이 가이드에서는 Red Hat OpenStack Platform Director를 사용하여 Red Hat OpenStack Platform 엔터프라이즈 환경의 특정 고급 기능을 구성하는 방법을 설명합니다. 여기에는 네트워크 격리, 스토리지 구성, SSL 통신 및 일반 구성 방법과 같은 기능이 포함됩니다.

차례

1장. 소개 **5**

2장. HEAT 템플릿 이해 **6**
2.1. HEAT 템플릿 6
2.2. 환경 파일 7
2.3. 코어 오버클라우드 HEAT 템플릿 8
2.4. 계획 환경 메타데이터 9
2.5. 기능 맵 10
2.6. 오버클라우드 생성 시 환경 파일 포함 11
2.7. 사용자 지정된 코어 HEAT 템플릿 사용 12
2.8. JINJA2 렌더링 15

3장. 매개 변수 **18**
3.1. 예 1: 시간대 구성 18
3.2. 예 2: 네트워킹 분산 가상 라우팅(DVR) 활성화 18
3.3. 예 3: RABBITMQ 파일 설명자 제한 구성 18
3.4. 예 4: 매개 변수 활성화 및 비활성화 19
3.5. 예 5: 역할 기반 매개변수 19
3.6. 수정할 매개 변수 식별 19

4장. 구성 후크 **22**
4.1. 첫 번째 부팅: 첫 번째 부팅 구성 사용자 정의 22
4.2. 구성 전: 특정 오버클라우드 역할 사용자 정의 23
4.3. 구성 전: 모든 오버클라우드 역할 사용자 정의 26
4.4. 구성 후: 모든 오버클라우드 역할 사용자 정의 28
4.5. PUPPET: 역할에 맞는 HIERADATA 사용자 정의 31
4.6. PUPPET: 개별 노드에 대한 HIERADATA 사용자 정의 32
4.7. PUPPET: 사용자 정의 매니페스트 적용 33

5장. 오버클라우드 등록 **35**
5.1. 환경 파일로 오버클라우드 등록 35
5.2. 예 1: 고객 포털에 등록 38
5.3. 예 2: RED HAT SATELLITE 6 SERVER에 등록 38
5.4. 예 3: RED HAT SATELLITE 5 SERVER에 등록 39
5.5. 예 4: HTTP 프록시를 통한 등록 40
5.6. 고급 등록 방법 40

6장. ANSIBLE 기반 오버클라우드 등록 **43**
6.1. RHSM(RED HAT SUBSCRIPTION MANAGER) 구성 가능 서비스 43
6.2. RHSMVARS 하위 매개변수 44
6.3. RHSM 구성 가능 서비스를 사용하여 오버클라우드 등록 44
6.4. 다른 역할에 RHSM 구성 가능 서비스 적용 45
6.5. RED HAT SATELLITE SERVER에 오버클라우드 등록 47
6.6. RHSM 구성 가능 서비스로 전환 47
6.7. RHEL-REGISTRATION TO RHSM MAPPINGS 48
6.8. RHSM 구성 가능 서비스를 사용하여 오버클라우드 배포 49

7장. 구성 가능 서비스 및 사용자 지정 역할 **51**
7.1. 지원되는 역할 아키텍처 51
7.2. 역할 51
7.3. 구성 가능 서비스 61

8장. 컨테이너화된 서비스 **68**

8.1. 컨테이너화된 서비스 아키텍처 68
8.2. 컨테이너화된 서비스 매개변수 69
8.3. OPENSTACK PLATFORM 컨테이너 수정 70
8.4. 벤더 플러그인 배포 72

9장. 기본 네트워크 격리 73
9.1. 네트워크 격리 73
9.2. 격리된 네트워크 구성 수정 75
9.3. 네트워크 인터페이스 템플릿 76
9.4. 기본 네트워크 인터페이스 템플릿 77
9.5. 기본 네트워크 격리 활성화 78

10장. 사용자 지정 구성 가능 네트워크 80
10.1. 구성 가능 네트워크 81
10.2. 구성 가능 네트워크 추가 81
10.3. 역할에 구성 가능한 네트워크 포함 82
10.4. 구성 가능한 네트워크에 OPENSTACK 서비스 할당 83
10.5. 사용자 지정 구성 가능 네트워크 활성화 84
10.6. 기본 네트워크 이름 변경 85

11장. 사용자 정의 네트워크 인터페이스 템플릿 87
11.1. 사용자 지정 네트워크 아키텍처 88
11.2. 사용자 정의를 위한 기본 네트워크 인터페이스 템플릿 렌더링 89
11.3. 네트워크 인터페이스 아키텍처 89
11.4. 네트워크 인터페이스 참조 90
11.5. 네트워크 인터페이스 레이아웃 예 100
11.6. 사용자 지정 네트워크의 네트워크 인터페이스 템플릿 고려 사항 102
11.7. 사용자 지정 네트워크 환경 파일 104
11.8. 네트워크 환경 매개변수 104
11.9. 사용자 지정 네트워크 환경 파일 예 108
11.10. 사용자 정의 NIC를 통한 네트워크 격리 활성화 108

12장. 추가 네트워크 구성 110
12.1. 사용자 정의 인터페이스 구성 110
12.2. 경로 및 기본 경로 구성 112
12.3. 점보 프레임 구성 112
12.4. 트렁크된 인터페이스에서 기본 VLAN 구성 114
12.5. NETFILTER가 추적하는 최대 연결 수 증가 114

13장. 네트워크 인터페이스 연결 118
13.1. LACP(NETWORK INTERFACE BONDING AND LINK AGGREGATION CONTROL PROTOCOL) 118
13.2. OPEN VSWITCH 본딩 옵션 120
13.3. LINUX 본딩 옵션 121
13.4. OVS 본딩 옵션 122

14장. 노드 배치 제어 124
14.1. 특정 노드 ID 할당 124
14.2. 사용자 지정 호스트 이름 할당 125
14.3. 예측 가능한 IP 할당 126
14.4. 예측 가능한 가상 IP 할당 129

15장. 오버클라우드 공개 엔드 포인트에서 SSL/TLS 활성화 130
15.1. 서명 호스트 초기화 130
15.2. 인증 기관 생성 130
15.3. 클라이언트에 인증 기관 추가 131

15.4. SSL/TLS 키 생성 131
15.5. SSL/TLS 인증서 서명 요청 생성 131
15.6. SSL/TLS 인증서 생성 133
15.7. SSL/TLS 활성화 133
15.8. 루트 인증서 삽입 135
15.9. DNS 끝점 구성 136
15.10. 오버클라우드 생성 중 환경 파일 추가 137
15.11. SSL/TLS 인증서 업데이트 138

16장. ID 관리를 사용하여 내부 및 공개 끝점에서 SSL/TLS 활성화 139
16.1. 언더클라우드를 CA에 추가합니다 139
16.2. IDM에 언더클라우드 추가 139
16.3. 오버클라우드 DNS 설정 141
16.4. NOVAJOIN을 사용하도록 오버클라우드 설정 142

17장. TLS를 사용하도록 기존 배포 변환 145
17.1. 요구 사항 145
17.2. 끝점 검토 145
17.3. 알려진 문제에 대한 해결 방법 적용 146
17.4. TLS를 사용하도록 끝점 구성 147
17.5. TLS 암호화 확인 157

18장. 디버그 모드 159

19장. 스토리지 설정 160
19.1. NFS 스토리지 구성 160
19.2. CEPH STORAGE 구성 163
19.3. 외부 오브젝트 스토리지 클러스터 사용 164
19.4. 이미지 가져오기 방법 및 공유 스테이징 영역 구성 165
19.5. 이미지 서비스의 CINDER 백엔드 구성 166
19.6. 하나의 인스턴스에 연결할 최대 스토리지 장치 수 구성 167
19.7. 타사 스토리지 구성 168

20장. 보안 개선 사항 170
20.1. 오버클라우드 방화벽 관리 170
20.2. SNMP(SIMPLE NETWORK MANAGEMENT PROTOCOL) 문자열 변경 172
20.3. HAPROXY에 대한 SSL/TLS 암호화 방식 및 규칙 변경 173
20.4. OPEN VSWITCH 방화벽 사용 175
20.5. 보안 루트 사용자 액세스 사용 175

21장. 네트워크 플러그인 구성 178
21.1. FUJITSU CONVERGED FABRIC (C-FABRIC) 178
21.2. FUJITSU FOS SWITCH 179

22장. ID 구성 181
22.1. 지역 이름 181

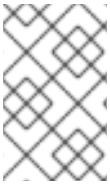
23장. 기타 구성 182
23.1. 오버클라우드 노드에서 커널 구성 182
23.2. 외부 로드 밸런싱 구성 182
23.3. IPV6 네트워킹 구성 183

1장. 소개

Red Hat OpenStack Platform director는 Overcloud라고도 하는 완전한 기능을 갖춘 OpenStack 환경을 프로비저닝하고 생성하는 툴을 제공합니다. Director 설치 및 사용 가이드 에서는 Overcloud 준비 및 구성에 대해 다룹니다. 그러나 적절한 프로덕션 수준 오버클라우드에는 다음을 포함한 추가 구성이 필요할 수 있습니다.

- 오버클라우드를 기존 네트워크 인프라에 통합하기 위한 기본 네트워크 구성.
- 특정 OpenStack 네트워크 트래픽 유형의 별도의 VLAN에서 네트워크 트래픽 격리.
- 공용 엔드 포인트에서 통신을 보호하기 위한 SSL 구성
- NFS, iSCSI, Red Hat Ceph Storage 및 여러 타사 스토리지 장치와 같은 스토리지 옵션.
- Red Hat Content Delivery Network 또는 내부 Red Hat Satellite 5 또는 6 서버에 노드 등록.
- 다양한 시스템 수준 옵션.
- 다양한 OpenStack 서비스 옵션.

이 가이드에서는 director를 통해 오버클라우드를 보강하는 방법을 설명합니다. 이때 director는 노드를 등록하고 Overcloud 생성에 필요한 서비스를 구성했습니다. 이제 이 가이드의 방법을 사용하여 오버클라우드를 사용자 지정할 수 있습니다.



참고

이 가이드의 예제는 Overcloud 구성을 위한 선택적 단계입니다. 이러한 단계는 Overcloud에 추가 기능을 제공하는 데만 필요합니다. 사용자 환경의 요구 사항에 적용되는 단계만 사용합니다.

2장. HEAT 템플릿 이해

이 가이드의 사용자 지정 구성은 Heat 템플릿 및 환경 파일을 사용하여 Overcloud의 특정 측면을 정의합니다. 이 장에서는 Heat 템플릿에 대한 기본적인 소개를 통해 Red Hat OpenStack Platform director의 맥락에서 이러한 템플릿의 구조와 형식을 이해할 수 있습니다.

2.1. HEAT 템플릿

RHOSP(Red Hat OpenStack Platform) director는 HOT(Heat Orchestration Templates)를 오버클라우드 배포 계획의 템플릿 형식으로 사용합니다. HOT 형식의 템플릿은 일반적으로 YAML 형식으로 표시됩니다. 템플릿의 목적은 heat에서 생성하는 리소스 컬렉션과 리소스의 구성인 *스택*을 정의하고 생성하는 것입니다. 리소스는 RHOSP의 개체이며 컴퓨팅 리소스, 네트워크 구성, 보안 그룹, 확장 규칙 및 사용자 지정 리소스를 포함할 수 있습니다.

참고

RHOSP에서 heat 템플릿 파일을 사용자 지정 템플릿 리소스로 사용하려면 파일 확장자가 **.yaml 또는. template** 이어야 합니다.

Heat 템플릿에는 세 가지 주요 섹션이 있습니다.

매개 변수

이러한 설정은 heat로 전달되어 스택을 사용자 지정합니다. heat 매개변수를 사용하여 기본값을 사용자 지정할 수도 있습니다. 이러한 설정은 템플릿의 **parameters** 섹션에 정의되어 있습니다.

Resources

이는 스택의 일부로 만들고 구성할 특정 오브젝트입니다. RHOSP(Red Hat OpenStack Platform)에는 모든 구성 요소에 걸쳐 있는 핵심 리소스 세트가 포함되어 있습니다. 이러한 값은 템플릿의 **resources** 섹션에 정의되어 있습니다.

출력 결과

이는 스택을 만든 후 heat에서 전달되는 값입니다. heat API 또는 클라이언트 도구를 통해 이러한 값에 액세스할 수 있습니다. 이러한 값은 템플릿의 **output** 섹션에 정의되어 있습니다.

다음은 기본 Heat 템플릿의 예입니다.

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

parameters:
  key_name:
    type: string
    default: lars
    description: Name of an existing key pair to use for the instance
  flavor:
    type: string
    description: Instance type for the instance to be created
    default: m1.small
  image:
    type: string
    default: cirros
    description: ID or name of the image to use for the instance
```

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      name: My Cirros Instance
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }

output:
  instance_name:
    description: Get the instance's name
    value: { get_attr: [ my_instance, name ] }
```

이 템플릿은 리소스 유형 **유형을 사용합니다. OS::Nova::Server** 특정 플레이버, 이미지 및 키를 사용하여 **my_instance** 라는 인스턴스를 만듭니다. 스택은 **My Cirros Instance** 라는 **instance_name** 값을 반환할 수 있습니다.

Heat에서 템플릿을 처리하면 템플릿의 스택과 리소스 템플릿의 하위 스택 세트를 생성합니다. 이렇게 하면 템플릿으로 정의한 기본 스택에서 내림되는 스택의 계층 구조가 생성됩니다. 다음 명령을 사용하여 스택 계층 구조를 볼 수 있습니다.

```
$ openstack stack list --nested
```

2.2. 환경 파일

환경 파일은 heat 템플릿에 대한 사용자 지정을 제공하는 특수 유형의 템플릿입니다. 여기에는 다음 세 가지 주요 부분이 포함됩니다.

리소스 레지스트리

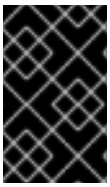
이 섹션에서는 다른 heat 템플릿에 연결된 사용자 지정 리소스 이름을 정의합니다. 이를 통해 핵심 리소스 컬렉션 내에 존재하지 않는 사용자 지정 리소스를 생성할 수 있습니다. 이러한 값은 환경 파일의 **resource_registry** 섹션에 정의되어 있습니다.

매개 변수

이는 최상위 템플릿의 매개 변수에 적용하는 일반적인 설정입니다. 예를 들어 리소스 레지스트리 매핑과 같이 중첩 스택을 배포하는 템플릿이 있는 경우 매개 변수는 중첩된 리소스에 대한 템플릿은 아닌 최상위 템플릿에만 적용됩니다. 매개 변수는 환경 파일의 **parameters** 섹션에 정의됩니다.

매개변수 기본값

이러한 매개 변수는 모든 템플릿에서 매개 변수의 기본값을 수정합니다. 예를 들어 리소스 레지스트리 매핑과 같이 중첩된 스택을 배포하는 heat 템플릿이 있는 경우 매개변수 기본값은 모든 템플릿에 적용됩니다. 매개 변수 기본값은 환경 파일의 **parameter_defaults** 섹션에 정의됩니다.



중요

오버클라우드에 대한 사용자 지정 환경 파일을 생성할 때 **매개변수** 대신 **parameter_defaults** 를 사용합니다. 따라서 매개 변수가 Overcloud의 모든 스택 템플릿에 적용됩니다.

기본 환경 파일의 예:

```
resource_registry:
  OS::Nova::Server::MyServer: myserver.yaml
```

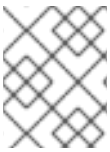
```
parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

heat 템플릿**my_ template.yaml에서 스택을 생성할 때 환경 파일 my_env.yaml** 이 포함될 수 있습니다. **my_env.yaml** 파일은 **OS::Nova::Server::MyServer** 라는 새 리소스 유형을 생성합니다. **myserver.yaml** 파일은 내장된 모든 리소스 유형을 재정의하는 이 리소스 유형의 구현을 제공하는 heat 템플릿 파일입니다. **my_template.yaml** 파일에 **OS::Nova::Server::MyServer** 리소스를 포함할 수 있습니다.

MyIP 는 이 환경 파일로 배포하는 기본 heat 템플릿에만 매개 변수를 적용합니다. 이 예제에서는 **my_template.yaml** 의 매개변수에만 적용됩니다.

NetworkName 은 기본 heat 템플릿 **my_template.yaml** 과 기본 템플릿(예 **: OS::Nova::Server::MyServer** 리소스 및 **myserver.yaml** 템플릿)이 포함된 리소스와 연결된 템플릿 모두에 적용됩니다.



참고

RHOSP에서 heat 템플릿 파일을 사용자 지정 템플릿 리소스로 사용하려면 파일 확장자가 **.yaml 또는. template** 이어야 합니다.

2.3. 코어 오버클라우드 HEAT 템플릿

director에는 오버클라우드의 코어 heat 템플릿 컬렉션이 포함되어 있습니다. 이 컬렉션은 **/usr/share/openstack-tripleo-heat-templates** 에 저장됩니다.

이 템플릿 컬렉션의 기본 파일과 디렉터리는 다음과 같습니다.

overcloud.j2.yaml

이 파일은 Overcloud 환경을 생성하는 기본 템플릿 파일입니다. 이 파일은 Jinja2 구문을 사용하여 템플릿의 특정 섹션을 반복하여 사용자 지정 역할을 생성합니다. Jinja2 형식은 Overcloud 배포 프로세스 중에 YAML로 렌더링됩니다.

overcloud-resource-registry-puppet.j2.yaml

이 파일은 Overcloud 환경을 생성하는 기본 환경 파일입니다. Overcloud 이미지에 저장된 Puppet 모듈에 대한 구성 세트를 제공합니다. director가 각 노드에 오버클라우드 이미지를 쓰고 나면 heat는 이 환경 파일에 등록된 리소스를 사용하여 각 노드의 Puppet 구성을 시작합니다. 이 파일은 Jinja2 구문을 사용하여 템플릿의 특정 섹션을 반복하여 사용자 지정 역할을 생성합니다. Jinja2 형식은 Overcloud 배포 프로세스 중에 YAML로 렌더링됩니다.

roles_data.yaml

이 파일은 Overcloud에서 역할을 정의하고 서비스를 각 역할에 매핑하는 파일입니다.

network_data.yaml

이 파일은 오버클라우드의 네트워크와 서브넷, 할당 풀, VIP 상태 등의 속성을 정의하는 파일입니다. 기본 **network_data** 파일에는 기본 네트워크가 포함되어 있습니다. 외부, 내부 API, 스토리지, 스토리지 관리, 테넌트 및 관리. 사용자 지정 **network_data** 파일을 생성하고 **-n** 옵션을 사용하여 **openstack overcloud deploy** 명령에 추가할 수 있습니다.

plan-environment.yaml

이 파일은 오버클라우드 계획의 메타데이터를 정의하는 파일입니다. 여기에는 계획 이름, 사용할 기본 템플릿, Overcloud에 적용할 환경 파일이 포함됩니다.

capabilities-map.yaml

이는 오버클라우드 계획에 대한 환경 파일 매핑입니다. 이 파일을 사용하여 Director web UI에서 환경 파일을 설명하고 활성화합니다. Overcloud 계획의 **환경** 디렉터리에서 감지되지만 **capabilities-map.yaml에 정의되지 않은 사용자 지정 환경 파일은 2의 기타 하위 탭에 나열되어 있습니다. 2 배포 구성 > 웹 UI의 전체 설정 지정**.

환경

오버클라우드 생성에 사용할 수 있는 추가 heat 환경 파일이 포함되어 있습니다. 이러한 환경 파일을 사용하면 결과 RHOSP(Red Hat OpenStack Platform) 환경에 추가 기능을 사용할 수 있습니다. 예를 들어 디렉터리에는 Cinder NetApp 백엔드 스토리지(cinder**-netapp-config.yaml)를 활성화하기 위한 환경 파일이 포함되어 있습니다. capabilities-map.yaml 파일에 정의되지 않은 이 디렉터리에서 감지된 환경 파일은 2의 기타 하위 탭에 나열되어 있습니다. 2 director의 웹 UI에서 배포 구성 > 전체 설정 지정**.

network

격리된 네트워크 및 포트를 만드는 데 도움이 되는 heat 템플릿 세트입니다.

Puppet

대부분 Puppet을 사용한 구성으로 구동되는 템플릿입니다. **overcloud-resource-registry-puppet.j2.yaml** 환경 파일은 이 디렉터리의 파일을 사용하여 각 노드에서 Puppet 구성의 애플리케이션을 구동합니다.

puppet/services

이 디렉터리는 구성 가능 서비스 아키텍처의 모든 서비스에 대한 heat 템플릿이 포함된 디렉터리입니다.

extraconfig

추가 기능을 활성화하는 템플릿입니다.

firstboot

director가 처음에 노드를 생성할 때 사용하는 **first_boot** 스크립트 예제를 제공합니다.

2.4. 계획 환경 메타데이터

계획 환경 메타데이터 파일을 사용하면 오버클라우드 계획에 대한 메타데이터를 정의할 수 있습니다. 이 정보는 오버클라우드 플랜을 가져오고 내보낼 때 사용되며 플랜에서 오버클라우드 생성 중에 사용됩니다.

계획 환경 메타데이터 파일에는 다음과 같은 매개변수가 포함되어 있습니다.

버전

템플릿의 버전입니다.

name

계획 파일을 저장하는 데 사용되는 OpenStack Object Storage(swift)의 Overcloud 계획 및 컨테이너 이름입니다.

template

Overcloud 배포에 사용할 코어 상위 템플릿입니다. 이는 **overcloud.yaml. j2 템플릿의 렌더링된 버전인 overcloud.yaml** 입니다.

환경

사용할 환경 파일 목록을 정의합니다. path 하위 매개 변수를 사용하여 각 환경 파일의 **경로를** 지정합니다.

parameter_defaults

오버클라우드에서 사용할 매개변수 세트입니다. 이 기능은 표준 환경 파일의 **parameter_defaults** 섹션과 동일한 방식으로 작동합니다.

암호

Overcloud 암호에 사용할 매개 변수 세트입니다. 이 기능은 표준 환경 파일의 **parameter_defaults** 섹션과 동일한 방식으로 작동합니다. 일반적으로 director는 임의로 생성된 암호로 이 섹션을 자동으로 채웁니다.

workflow_parameters

OpenStack Workflow(mistral) 네임스페이스에 매개 변수 집합을 제공할 수 있습니다. 이를 사용하여 특정 오버클라우드 매개변수를 계산하고 자동으로 생성할 수 있습니다.

다음은 계획 환경 파일의 구문 예입니다.

```
version: 1.0
name: myovercloud
description: 'My Overcloud Plan'
template: overcloud.yaml
environments:
- path: overcloud-resource-registry-puppet.yaml
- path: environments/docker.yaml
- path: environments/docker-ha.yaml
- path: environments/containers-default-parameters.yaml
- path: user-environment.yaml
parameter_defaults:
  ControllerCount: 1
  ComputeCount: 1
  OvercloudComputeFlavor: compute
  OvercloudControllerFlavor: control
workflow_parameters:
  tripleo.derive_params.v1.derive_parameters:
    num_phy_cores_per_numa_node_for_pmd: 2
```

-p 옵션을 사용하여 **openstack overcloud deploy** 명령을 사용하여 계획 환경 메타데이터 파일을 포함할 수 있습니다. 예를 들면 다음과 같습니다.

```
(undercloud) $ openstack overcloud deploy --templates \
  -p /my-plan-environment.yaml \
  [OTHER OPTIONS]
```

다음 명령을 사용하여 기존 오버클라우드 계획의 계획 메타데이터를 볼 수도 있습니다.

```
(undercloud) $ openstack object save overcloud plan-environment.yaml --file -
```

2.5. 기능 맵

기능 맵은 계획 및 해당 종속 항목의 환경 파일 매핑을 제공합니다. 이 파일을 사용하여 director의 웹 UI를 통해 환경 파일을 설명하고 활성화합니다. Overcloud 계획에서 감지되지만 **capabilities-map.yaml** 에 나열되지 않은 사용자 지정 환경 파일은 2 **배포 구성 > 웹 UI의 전체 설정 지정 2**의 **기타** 하위 탭에 나열되어 있습니다.

기본 파일은 **/usr/share/openstack-tripleo-heat-templates/capabilities-map.yaml** 에 있습니다.

다음은 기능 맵의 구문 예입니다.

```
topics: 1
  - title: My Parent Section
    description: This contains a main section for different environment files
```

```
environment_groups: (2)
  - name: my-environment-group
    title: My Environment Group
    description: A list of environment files grouped together
    environments: (3)
      - file: environment_file_1.yaml
        title: Environment File 1
        description: Enables environment file 1
        requires: (4)
          - dependent_environment_file.yaml
      - file: environment_file_2.yaml
        title: Environment File 2
        description: Enables environment file 2
        requires: (5)
          - dependent_environment_file.yaml
      - file: dependent_environment_file.yaml
        title: Dependent Environment File
        description: Enables the dependent environment file
```

(1) **topics** 매개 변수에는 UI 배포 구성에 섹션 목록이 포함되어 있습니다. 각 주제는 단일 환경 옵션 화면으로 표시되며, environment **_groups 매개 변수로 정의된 여러 환경** 그룹이 포함되어 있습니다. 각 항목에는 일반 텍스트 **제목과 설명이** 있을 수 있습니다.

(2) **environment_groups** 매개 변수는 UI의 배포 구성에 있는 환경 파일 그룹을 나열합니다. 예를 들어 스토리지 주제에서 Ceph 관련 환경 파일을 위한 환경 그룹이 있을 수 있습니다. 각 환경 그룹에는 일반 텍스트 **제목과 설명이** 있을 수 있습니다.

(3) **environments** 매개 변수는 환경 그룹에 속하는 모든 환경 파일을 보여줍니다. **file** 매개 변수는 환경 파일의 위치입니다. 각 환경 항목에는 일반 텍스트 **제목과 설명이** 있을 수 있습니다.

(4) (5) **requires** 매개 변수는 환경 파일의 종속성 목록입니다. 이 예에서 **environment_file_1.yaml** 및 **environment_file_2.yaml** 모두 **dependent_environment_file.yaml** 도 활성화해야 합니다.

참고

Red Hat OpenStack Platform은 이 파일을 사용하여 director UI에 기능에 대한 액세스를 추가합니다. 최신 버전의 Red Hat OpenStack Platform이 이 파일을 재정의할 수 있으므로 이 파일을 수정하지 않는 것이 좋습니다.

2.6. 오버클라우드 생성 시 환경 파일 포함

배포 명령(**openstack overcloud deploy**)은 **-e** 옵션을 사용하여 환경 파일을 포함하여 오버클라우드를 사용자 지정합니다. 환경 파일은 필요한 수만큼 추가할 수 있습니다. 차후에 실행되는 환경 파일에 정의된 매개변수와 리소스가 우선순위를 갖기 때문에 환경 파일의 순서가 중요합니다. 예를 들어 다음 두 개의 환경 파일이 있을 수 있습니다.

environment-file-1.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-1.yaml

parameter_defaults:
  RabbitFDLimit: 65536
  TimeZone: 'Japan'
```

-

environment-file-2.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-2.yaml

parameter_defaults:
  TimeZone: 'Hongkong'
```

그런 다음 두 환경 파일을 모두 포함하여 배포합니다.

```
$ openstack overcloud deploy --templates -e environment-file-1.yaml -e environment-file-2.yaml
```

이 예에서 두 환경 파일에는 공통 리소스 유형(OS::**TripleO::NodeExtraConfigPost**)과 공통 매개 변수(Time**Zone**)가 포함됩니다. **openstack overcloud deploy** 명령은 다음 프로세스를 통해 실행됩니다.

1. **--template** 옵션에 따라 코어 Heat 템플릿 컬렉션에서 기본 구성을 로드합니다.
2. **environment-file-1.yaml** 의 구성을 적용하여 기본 구성의 일반적인 설정을 재정의합니다.
3. **environment-file-2.yaml** 의 구성을 적용하여 기본 구성 및 **environment-file-1.yaml** 의 공통 설정을 재정의합니다.

그러면 오버클라우드의 기본 구성이 다음과 같이 변경됩니다.

- **OS::TripleO::NodeExtraConfigPost** 리소스는 **environment-file-2.yaml에 따라 /home/stack/templates/template-2.yaml** 로 설정됩니다.
- **timezone 매개변수는 environment-file-2.yaml** 에 따라 **Hongkong** 으로 설정됩니다.
- **RabbitFDLimit** 매개변수는 **environment-file-1.yaml** 에 따라 **65536** 으로 설정됩니다.**environment-file-2.yaml** 은 이 값을 변경하지 않습니다.

이렇게 하면 여러 환경 파일 충돌의 값 없이 오버클라우드에 대한 사용자 지정 구성을 정의할 수 있습니다.

2.7. 사용자 지정된 코어 HEAT 템플릿 사용

오버클라우드를 생성할 때 director는 **/usr/share/openstack-tripleo-heat-templates** 에 있는 코어 Heat 템플릿 세트를 사용합니다. 이 핵심 템플릿 컬렉션을 사용자 지정하려면 Git 워크플로를 사용하여 변경 사항을 추적하고 업데이트를 병합합니다. 다음 git 프로세스를 사용하여 사용자 지정 템플릿 컬렉션을 관리합니다.

사용자 정의 템플릿 컬렉션 초기화

다음 절차에 따라 Heat 템플릿 컬렉션이 포함된 초기 Git 리포지토리를 생성합니다.

1. 템플릿 컬렉션을 **스택** 사용자 디렉터리에 복사합니다. 이 예제에서는 컬렉션을 **~/templates** 디렉터리에 복사합니다.

   ```
   $ cd ~/templates
   $ cp -r /usr/share/openstack-tripleo-heat-templates .
   ```

2. 사용자 지정 템플릿 디렉터리로 변경하고 Git 리포지토리를 초기화합니다.

```
$ cd openstack-tripleo-heat-templates
$ git init .
```

3. Git 사용자 이름 및 이메일 주소를 구성합니다.

```
$ git config --global user.name "<USER_NAME>"
$ git config --global user.email "<EMAIL_ADDRESS>"
```

<USER_NAME> 을 사용하려는 사용자 이름으로 바꿉니다. **<EMAIL_ADDRESS>** 를 이메일 주소로 바꿉니다.

4. 초기 커밋을 위한 모든 템플릿을 스테이징합니다.

```
$ git add *
```

5. 초기 커밋을 생성합니다.

```
$ git commit -m "Initial creation of custom core heat templates"
```

이렇게 하면 최신 코어 템플릿 컬렉션을 포함하는 초기 **master** 분기가 생성됩니다. 이 분기를 사용자 지정 분기의 기반으로 사용하고 새 템플릿 버전을 이 분기에 병합합니다.

사용자 정의 분기 생성 및 변경 사항 커밋

사용자 지정 분기를 사용하여 코어 템플릿 컬렉션에 변경 사항을 저장합니다. 다음 절차에 따라 **my-customizations** 분기를 생성하고 사용자 지정을 추가합니다.

1. **my-customizations** 분기를 생성하여 해당 분기로 전환합니다.

```
$ git checkout -b my-customizations
```

2. 사용자 지정 분기의 파일을 편집합니다.

3. git에서 변경 사항을 스테이징합니다.

```
$ git add [edited files]
```

4. 사용자 정의 분기의 변경 사항을 커밋합니다.

```
$ git commit -m "[Commit message for custom changes]"
```

그러면 **my-customizations** 분기에 대한 커밋으로 변경 사항이 추가됩니다. **master** 분기가 업데이트되면 마스터에서 **my-customizations** 를 다시 평가 **하여** git에서 이러한 커밋을 업데이트된 템플릿 컬렉션에 추가할 수 있습니다. 이렇게 하면 사용자 지정을 추적하고 향후 템플릿 업데이트에서 재생하는 데 도움이 됩니다.

사용자 정의 템플릿 컬렉션 업데이트:

언더클라우드를 업데이트할 때 **openstack-tripleo-heat-templates** 패키지도 업데이트할 수 있습니다. 이 경우 다음 절차를 사용하여 사용자 정의 템플릿 컬렉션을 업데이트합니다.

1. **openstack-tripleo-heat-templates** 패키지 버전을 환경 변수로 저장합니다.

```
$ export PACKAGE=$(rpm -qv openstack-tripleo-heat-templates)
```

2. 템플릿 컬렉션 디렉터리로 변경하고 업데이트된 템플릿의 새 분기를 생성합니다.

```
$ cd ~/templates/openstack-tripleo-heat-templates
$ git checkout -b $PACKAGE
```

3. 분기의 모든 파일을 제거하고 새 버전으로 바꿉니다.

```
$ git rm -rf *
$ cp -r /usr/share/openstack-tripleo-heat-templates/* .
```

4. 초기 커밋을 위한 모든 템플릿을 추가합니다.

```
$ git add *
```

5. 패키지 업데이트에 대한 커밋을 생성합니다.

```
$ git commit -m "Updates for $PACKAGE"
```

6. 분기를 master에 병합합니다. Git 관리 시스템(예: GitLab)을 사용하는 경우 관리 워크플로를 사용합니다. Git을 로컬로 사용하는 경우 **master** 분기로 전환하여 병합하고 **git merge** 명령을 실행합니다.

```
$ git checkout master
$ git merge $PACKAGE
```

이제 **master** 분기에 최신 버전의 코어 템플릿 컬렉션이 포함됩니다. 이제 이 업데이트된 컬렉션에서 **my-customization** 분기를 다시 평가할 수 있습니다.

사용자 정의 분기 변경

다음 절차를 사용하여 **my-customization** 분기를 업데이트합니다.

1. **my-customizations** 분기로 변경합니다.

```
$ git checkout my-customizations
```

2. **마스터에서** 분기를 다시베이스합니다.

```
$ git rebase master
```

그러면 **my-customizations** 분기를 업데이트하고 이 분기에 대한 사용자 지정 커밋을 재생합니다.

git가 리베이스 중에 충돌을 보고하는 경우 다음 절차를 사용하십시오.

1. 충돌이 포함된 파일을 확인합니다.

```
$ git status
```

2. 식별된 템플릿 파일의 충돌을 해결합니다.

3. 해결된 파일 추가

```
$ git add [resolved files]
```

4. 업데이트를 계속합니다.

```
$ git rebase --continue
```

사용자 정의 템플릿 배포

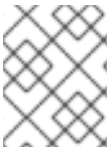
다음 절차를 사용하여 사용자 지정 템플릿 컬렉션을 배포합니다.

1. **my-customization** 분기로 전환되었는지 확인합니다.

```
git checkout my-customizations
```

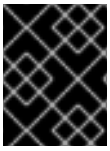
2. 로컬 템플릿 디렉터리를 지정하려면 **--templates** 옵션과 함께 **openstack overcloud deploy** 명령을 실행합니다.

```
$ openstack overcloud deploy --templates /home/stack/templates/openstack-tripleo-heat-templates [OTHER OPTIONS]
```



참고

디렉터리 없이 **--templates 옵션을 지정하는 경우 director는 기본 템플릿 디렉터리 (/usr/share/openstack-tripleo-heat-templates**)를 사용합니다.



중요

Red Hat은 heat 템플릿 컬렉션을 수정하는 대신 4장. *구성 후크* 에서 방법을 사용하는 것이 좋습니다.

2.8. JINJA2 렌더링

/usr/share/openstack-tripleo-heat-templates 의 코어 Heat 템플릿에는 **j2.yaml** 확장자로 끝나는 많은 파일이 포함되어 있습니다. 이러한 파일에는 Jinja2 템플릿 구문이 포함되어 있으며 director는 이러한 파일을 **.yaml** 로 끝나는 정적 Heat 템플릿에 렌더링합니다. 예를 들어 기본 **overcloud.j2.yaml** 파일은 **overcloud.yaml** 에 렌더링됩니다. director는 생성된 **overcloud.yaml** 파일을 사용합니다.

Jinja2 지원 Heat 템플릿은 Jinja2 구문을 사용하여 반복적인 값에 대한 매개 변수와 리소스를 생성합니다. 예를 들어 **overcloud.j2.yaml** 파일에는 다음 스니펫이 포함되어 있습니다.

```
parameters:
...
{% for role in roles %}
  ...
  {{role.name}}Count:
    description: Number of {{role.name}} nodes to deploy
    type: number
    default: {{role.CountDefault|default(0)}}
  ...
{% endfor %}
```

director가 Jinja2 구문을 렌더링하면 director가 **roles_data.yaml** 파일에 정의된 역할을 반복하고 **{{role.name}}Count** 매개 변수를 역할 이름으로 채웁니다. 기본 **roles_data.yaml** 파일에는 5개의 역할이 포함되어 있으며, 결과적으로 이 예제에서 다음 매개변수가 생성됩니다.

- **ControllerCount**

- **ComputeCount**
- **BlockStorageCount**
- **ObjectStorageCount**
- **CephStorageCount**

렌더링된 매개변수의 예는 다음과 같습니다.

```
parameters:
  ...
  ControllerCount:
    description: Number of Controller nodes to deploy
    type: number
    default: 1
  ...
```

director는 핵심 Heat 템플릿 디렉터리 내에서 Jinja2 지원 템플릿과 환경 파일만 렌더링합니다. 다음 사용 사례에서는 Jinja2 템플릿을 렌더링하는 올바른 방법을 보여줍니다.

사용 사례 1: 기본 코어 템플릿

template 디렉터리: /usr/share/openstack-tripleo-heat-templates/

환경 파일: /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.j2.yaml

director는 기본 코어 템플릿 위치(**--templates**)를 사용합니다. director는 **network-isolation.j2.yaml 파일을 network -isolation.yaml** 로 렌더링합니다. **openstack overcloud deploy** 명령을 실행하는 경우 **-e** 옵션을 사용하여 렌더링된 **network-isolation.yaml** 파일의 이름을 포함합니다.

```
$ openstack ovecloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml
    ...
```

사용 사례 2: 사용자 정의 코어 템플릿

템플릿 디렉터리: /home/stack/tripleo-heat-templates

환경 파일: /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml

director는 사용자 지정 코어 템플릿 위치(**--templates /home/stack/tripleo-heat-templates**)를 사용합니다. director는 사용자 지정 코어 템플릿 내에서 **network-isolation.j2.yaml** 파일을 **network-isolation.yaml** 에 렌더링합니다. **openstack overcloud deploy** 명령을 실행하는 경우 **-e** 옵션을 사용하여 렌더링된 **network-isolation.yaml** 파일의 이름을 포함합니다.

```
$ openstack ovecloud deploy --templates /home/stack/tripleo-heat-templates \
    -e /home/stack/tripleo-heat-templates/environments/network-isolation.yaml
    ...
```

활용 사례 3: 사용 올바르지 않음

template 디렉터리: /usr/share/openstack-tripleo-heat-templates/

환경 파일: /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml

이 director는 사용자 지정 코어 템플릿 위치(**--templates /home/stack/tripleo-heat-templates**)를 사용합니다. 그러나 선택한 **network-isolation.j2.yaml** 은 사용자 지정 코어 템플릿 내에 없으므로 **network-isolation.yaml** 로 렌더링 *되지 않습니다*. 이로 인해 배포가 실패합니다.

3장. 매개 변수

director의 템플릿 컬렉션에 있는 각 Heat 템플릿에는 **parameters** 섹션이 포함되어 있습니다. 이 섹션에서는 특정 Overcloud 서비스와 관련된 모든 매개 변수를 정의합니다. 여기에는 다음이 포함됩니다.

- **overcloud.j2.yaml** - 기본 기본 매개변수
- **roles_data.yaml** - 구성 가능 역할에 대한 기본 매개변수
- **Puppet/services/*.yaml** - 특정 서비스에 대한 기본 매개변수

다음 방법을 사용하여 이러한 매개변수의 값을 수정할 수 있습니다.

1. 사용자 지정 매개 변수에 대한 환경 파일을 만듭니다.
2. 사용자 지정 매개 변수를 환경 파일의 **parameter_defaults** 섹션에 포함합니다.
3. **openstack overcloud deploy** 명령을 사용하여 환경 파일을 포함합니다.

다음 몇 섹션에는 **puppet/services** 디렉터리의 서비스에 대한 특정 매개 변수를 구성하는 방법을 보여주는 예제가 포함되어 있습니다.

3.1. 예 1: 시간대 구성

시간대(puppet/**services/time/timezone.yaml**)를 설정하는 Heat 템플릿에는 **TimeZone** 매개변수가 포함되어 있습니다. **TimeZone** 매개변수를 비워 두면 오버클라우드에서 시간을 기본값으로 **UTC** 로 설정합니다.

시간대 목록을 가져오려면 **timedatectl list-timezones** 명령을 실행합니다. 다음 예제 명령은 아시아의 시간대를 검색합니다.

```
$ sudo timedatectl list-timezones|grep "Asia"
```

시간대를 식별한 후 환경 파일에서 *TimeZone* 매개변수를 설정합니다. 다음 예제 환경 파일은 *TimeZone* 값을 *Asia/Tokyo* 로 설정합니다.

```
parameter_defaults:
  TimeZone: 'Asia/Tokyo'
```

3.2. 예 2: 네트워킹 분산 가상 라우팅(DVR) 활성화

OpenStack Networking(neutron) API(puppet/**services/neutron-api.yaml**)에 대한 Heat 템플릿에는 DVR(Distributed Virtual Routing)을 활성화 및 비활성화하는 매개 변수가 포함되어 있습니다. 매개 변수의 기본값은 **false** 입니다. 그러나 환경 파일에서 다음을 사용하여 활성화할 수 있습니다.

```
parameter_defaults:
  NeutronEnableDVR: true
```

3.3. 예 3: RABBITMQ 파일 설명자 제한 구성

특정 구성의 경우 RabbitMQ 서버의 파일 설명자 제한을 늘려야 할 수 있습니다. **puppet/services/rabbitmq.yaml** Heat 템플릿을 사용하면 **RabbitFDLimit** 매개변수를 필요한 제한으로 설정할 수 있습니다. 환경 파일에 다음을 추가합니다.

```
parameter_defaults:
  RabbitFDLimit: 65536
```

3.4. 예 4: 매개 변수 활성화 및 비활성화

배포 중에 매개 변수를 초기에 설정한 다음 업데이트 또는 확장 작업과 같은 향후 배포 작업에 대한 매개 변수를 비활성화해야 하는 경우도 있습니다. 예를 들어 오버클라우드 생성 중에 사용자 지정 RPM을 포함하려면 다음을 포함합니다.

```
parameter_defaults:
  DeployArtifactURLs: ["http://www.example.com/myfile.rpm"]
```

이후 배포에서 이 매개변수를 비활성화해야 하는 경우 매개 변수를 제거하는 것만으로는 충분하지 않습니다. 대신 매개변수를 빈 값으로 설정합니다.

```
parameter_defaults:
  DeployArtifactURLs: []
```

이렇게 하면 후속 배포 작업에 더 이상 매개변수가 설정되지 않습니다.

3.5. 예 5: 역할 기반 매개변수

[ROLE]Parameters 매개 변수를 사용하여 **[ROLE]** 을 구성 가능 역할로 교체하여 특정 역할에 대한 매개 변수를 설정합니다.

예를 들어 director는 컨트롤러 및 컴퓨팅 노드 모두에 **logrotate** 를 구성합니다. 컨트롤러 및 컴퓨팅 노드에 대해 다른 DestinationRule 매개변수를 설정하려면 'ControllerParameters' 및 'ComputeParameters' 매개변수가 모두 포함된 환경 파일을 생성하고 각 특정 역할에 대해 DestinationRule 매개변수를 설정합니다.

```
parameter_defaults:
  ControllerParameters:
    LogrotateMaxsize: 10M
    LogrotatePurgeAfterDays: 30
  ComputeParameters:
    LogrotateMaxsize: 20M
    LogrotatePurgeAfterDays: 15
```

3.6. 수정할 매개 변수 식별

Red Hat OpenStack Platform director는 설정에 필요한 여러 매개변수를 제공합니다. 경우에 따라 설정하기 위한 특정 옵션과 해당 director 매개변수를 식별하는 데 어려움이 있을 수 있습니다. director를 통해 설정하려는 옵션이 있는 경우 다음 워크플로를 사용하여 옵션을 식별하고 특정 오버클라우드 매개변수에 매핑합니다.

1. **구성하려는 옵션을 식별합니다. 옵션을 사용하는 서비스를 기록합니다.**

2. 이 옵션에 해당하는 Puppet 모듈을 확인합니다. Red Hat OpenStack Platform용 Puppet 모듈은 director 노드의 /etc/puppet/modules 에 있습니다. 각 모듈은 특정 서비스에 해당합니다. 예를 들어 keystone 모듈은 OpenStack ID(keystone)에 해당합니다.

   - Puppet 모듈에 선택한 옵션을 제어하는 변수가 포함된 경우 다음 단계로 이동합니다.
   - Puppet 모듈에 선택한 옵션을 제어하는 변수가 없으면 이 옵션에 대한 hieradata가 없습니다. 가능한 경우 오버클라우드가 배포를 완료한 후 수동으로 옵션을 설정할 수 있습니다.

3. hieradata 형식으로 Puppet 변수의 director의 핵심 Heat 템플릿 컬렉션을 확인합니다. 일반적으로 puppet/services/* 의 템플릿은 동일한 서비스의 Puppet 모듈에 해당합니다. 예를 들어 puppet/services/keystone.yaml 템플릿은 keystone 모듈에 hieradata를 제공합니다.

   - Heat 템플릿에서 Puppet 변수의 hieradata를 설정하는 경우 템플릿에서 수정할 director 기반 매개 변수도 공개해야 합니다.
   - Heat 템플릿에서 Puppet 변수에 hieradata를 설정하지 않으면 구성 후크를 사용하여 환경 파일을 사용하여 hieradata를 전달합니다. hieradata 사용자 지정에 대한 자세한 내용은 4.5절. “Puppet: 역할에 맞는 Hieradata 사용자 정의” 을 참조하십시오.

중요

동일한 사용자 지정 hieradata 해시의 여러 인스턴스를 정의하지 마십시오. 동일한 사용자 지정 hieradata의 여러 인스턴스로 인해 Puppet 실행 중에 충돌하여 구성 옵션에 대해 예기치 않은 값이 설정될 수 있습니다.

워크플로우 예

OpenStack ID(keystone)에 대한 알림 형식을 변경할 수 있습니다. 워크플로를 사용하면 다음을 수행할 수 있습니다.

1. 구성할 OpenStack 매개 변수를 식별합니다(notification_format).
2. keystone Puppet 모듈을 검색하여 notification_format 설정을 검색합니다. 예를 들면 다음과 같습니다.

```
$ grep notification_format /etc/puppet/modules/keystone/manifests/*
```

이 경우 **keystone** 모듈은 **keystone ::notification_format** 변수를 사용하여 이 옵션을 관리합니다.

3. 이 변수에 대해 **keystone** 서비스 템플릿을 검색합니다. 예를 들면 다음과 같습니다.

```
$ grep "keystone::notification_format" /usr/share/openstack-tripleo-heat-templates/puppet/services/keystone.yaml
```

출력에는 **KeystoneNotificationFormat** 매개변수를 사용하여 **director**가 **keystone::notification_format hieradata**를 설정하는 것을 보여줍니다.

다음 표는 최종 매핑을 보여줍니다.

director 매개변수	Puppet Hieradata	OpenStack Identity(keystone) 옵션
KeystoneNotificationFormat	**keystone::notification_format**	**notification_format**

즉, **Overcloud** 환경 파일에서 **KeystoneNotificationFormat** 을 설정하면 **Overcloud** 구성 중에 **keystone.conf** 파일에 **notification_format** 옵션이 설정됩니다.

4장. 구성 후크

구성 후크는 **Overcloud** 배포 프로세스에 고유한 구성 기능을 삽입하는 방법을 제공합니다. 여기에는 기본 **Overcloud** 서비스 구성 전후에 사용자 정의 구성을 삽입하는 후크와 **Puppet** 기반 구성을 수정하고 포함하기 위한 후크가 포함됩니다.

4.1. 첫 번째 부팅: 첫 번째 부팅 구성 사용자 정의

director는 **Overcloud**를 처음 생성할 때 모든 노드에서 구성을 수행하는 메커니즘을 제공합니다. **director**는 **OS::TripleO::NodeUserData** 리소스 유형을 사용하여 호출할 수 있는 **cloud-init** 를 통해 이 작업을 수행합니다.

이 예제에서는 모든 노드의 사용자 지정 **IP** 주소로 이름 서버를 업데이트합니다. 먼저 스크립트를 실행하여 각 노드의 **resolv.conf**를 특정 이름 서버에 추가하는 기본 **heat** 템플릿 (**/home/stack/templates/nameserver.yaml**)을 생성해야 합니다. **OS::TripleO::MultipartMime** 리소스 유형을 사용하여 구성 스크립트를 보낼 수 있습니다.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: nameserver_config}

  nameserver_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
        #!/bin/bash
        echo "nameserver 192.168.1.1" >> /etc/resolv.conf

outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

다음으로 **heat** 템플릿을 **OS::TripleO::NodeUserData** 리소스 유형으로 등록하는 환경 파일 (**/home/stack/templates/firstboot.yaml**)을 만듭니다.

```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

첫 번째 부팅 구성을 추가하려면 **Overcloud**를 처음 생성할 때 다른 환경 파일과 함께 스택에 환경 파일을 추가합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  ...
  -e /home/stack/templates/firstboot.yaml \
  ...
```

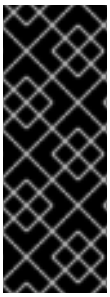
-e 는 오버클라우드 스택에 환경 파일을 적용합니다.

그러면 처음 생성될 때 모든 노드에 구성이 추가되고 처음으로 부팅됩니다. 이후 오버클라우드 스택 업데이트와 같은 이러한 템플릿은 이러한 스크립트를 실행하지 않습니다.

중요

OS::TripleO::NodeUserData 를 하나의 **heat** 템플릿에만 등록할 수 있습니다. 후속 사용은 사용할 **heat** 템플릿을 재정의합니다.

4.2. 구성 전: 특정 오버클라우드 역할 사용자 정의



중요

이 문서의 이전 버전에서는 **OS::TripleO::Tasks::*PreConfig** 리소스를 사용하여 역할별로 사전 구성 후크를 제공했습니다. **director**의 **Heat** 템플릿 컬렉션에는 이러한 후크를 전용으로 사용해야 하므로 사용자 지정 용도로 사용해서는 안 됩니다. 대신 아래에 설명된 **OS::TripleO::*ExtraConfigPre** 후크를 사용합니다.

Overcloud는 **OpenStack** 구성 요소의 핵심 구성에 **Puppet**을 사용합니다. **director**는 첫 번째 부팅이 완료된 후 코어 구성이 시작되기 전에 특정 노드 역할에 대한 사용자 정의 구성을 제공하는 후크 세트를 제공합니다. 이러한 후크에는 다음이 포함됩니다.

OS::TripleO::ControllerExtraConfigPre

핵심 **Puppet** 구성 전에 컨트롤러 노드에 적용된 추가 구성입니다.

OS::TripleO::ComputeExtraConfigPre

핵심 **Puppet** 구성 전에 컴퓨팅 노드에 추가 구성이 적용됩니다.

OS::TripleO::CephStorageExtraConfigPre

핵심 **Puppet** 구성 전에 **Ceph Storage** 노드에 추가 구성이 적용됩니다.

OS::TripleO::ObjectStorageExtraConfigPre

코어 **Puppet** 구성 전에 **Object Storage** 노드에 적용된 추가 구성입니다.

OS::TripleO::BlockStorageExtraConfigPre

핵심 **Puppet** 구성 전에 블록 스토리지 노드에 추가 구성이 적용됩니다.

OS::TripleO::[ROLE]ExtraConfigPre

코어 **Puppet** 구성 전에 사용자 지정 노드에 추가 구성이 적용됩니다. **[ROLE]** 을 구성 가능 역할 이름으로 교체합니다.

이 예에서 먼저 변수 이름 서버를 사용하여 스크립트를 실행하여 노드의 **resolv.conf**에 쓰는 스크립트를 실행하는 기본 **heat** 템플릿(**/home/stack/templates/nameserver.yaml**)을 생성합니다.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" > /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE','UPDATE']
```

```
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

  outputs:
    deploy_stdout:
      description: Deployment reference, used to trigger pre-deploy on changes
      value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

이 예제에서 **resources** 섹션에는 다음이 포함됩니다.

CustomExtraConfigPre

소프트웨어 구성을 정의합니다. 이 예제에서는 **Bash** 스크립트를 정의하고 **Heat**에서 **_NAMESERVER_IP_** 를 **nameserver_ip** 매개 변수에 저장된 값으로 바꿉니다.

CustomExtraDeploymentPre

그러면 **CustomExtraConfigPre** 리소스의 소프트웨어 구성인 소프트웨어 구성이 실행됩니다. 다음을 확인합니다.

- **config** 매개변수는 **CustomExtraConfigPre** 리소스에 대한 참조를 만들어 **Heat**에서 적용할 구성을 알 수 있습니다.
- **server** 매개 변수는 **Overcloud** 노드 맵을 검색합니다. 이 매개 변수는 상위 템플릿에서 제공하며 이 후크의 템플릿에서 필요합니다.
- **actions** 매개 변수는 구성을 적용할 시기를 정의합니다. 이 경우 오버클라우드를 생성하거나 업데이트하는 경우에만 구성을 적용합니다. 가능한 작업에는 **CREATE,UPDATE,DELETE,SUSPEND, RESUME** 등이 있습니다.
- **input_values** 에는 상위 템플릿에서 **DeployIdentifier** 를 저장하는 **deploy_identifier** 라는 매개 변수가 포함되어 있습니다. 이 매개 변수는 각 배포 업데이트의 리소스에 타임스탬프를 제공합니다. 이렇게 하면 리소스가 후속 오버클라우드 업데이트에 다시 적용됩니다.

다음으로 **heat** 템플릿을 역할 기반 리소스 유형에 등록하는 환경 파일 (**/home/stack/templates/pre_config.yaml**)을 생성합니다. 예를 들어 컨트롤러 노드에만 적용하려면 **ControllerExtraConfigPre** 후크를 사용합니다.

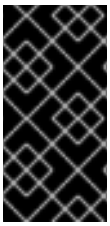
```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml
```

```
parameter_defaults:
  nameserver_ip: 192.168.1.1
```

구성을 적용하려면 **Overcloud**를 생성하거나 업데이트할 때 다른 환경 파일과 함께 스택에 환경 파일을 추가합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

이는 초기 **Overcloud** 생성 또는 후속 업데이트에서 코어 구성이 시작되기 전에 모든 컨트롤러 노드에 구성을 적용합니다.



중요

각 리소스를 후크당 하나의 **Heat** 템플릿에만 등록할 수 있습니다. 후속 사용량은 사용할 **Heat** 템플릿을 재정의합니다.

4.3. 구성 전: 모든 오버클라우드 역할 사용자 정의

Overcloud는 **OpenStack** 구성 요소의 핵심 구성에 **Puppet**을 사용합니다. **director**는 첫 번째 부팅이 완료된 후 코어 구성이 시작되기 전에 모든 노드 유형을 구성하는 후크를 제공합니다.

OS::TripleO::NodeExtraConfig

핵심 **Puppet** 구성 전에 모든 노드 역할에 적용되는 추가 구성입니다.

이 예제에서는 먼저 스크립트를 실행하여 각 노드의 **resolv.conf**를 변수 이름 서버로 추가하는 기본 **heat** 템플릿(**/home/stack/templates/nameserver.yaml**)을 생성합니다.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string
```

```
resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE','UPDATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

이 예제에서 **resources** 섹션에는 다음이 포함됩니다.

CustomExtraConfigPre

소프트웨어 구성을 정의합니다. 이 예제에서는 **Bash** 스크립트를 정의하고 **Heat**에서 **_NAMESERVER_IP_** 를 **nameserver_ip** 매개 변수에 저장된 값으로 바꿉니다.

CustomExtraDeploymentPre

그러면 **CustomExtraConfigPre** 리소스의 소프트웨어 구성인 소프트웨어 구성이 실행됩니다. 다음을 확인합니다.

- **config** 매개변수는 **CustomExtraConfigPre** 리소스에 대한 참조를 만들어 **Heat**에서 적용할 구성을 알 수 있습니다.
- **server** 매개 변수는 **Overcloud** 노드 맵을 검색합니다. 이 매개 변수는 상위 템플릿에서 제공하며 이 후크의 템플릿에서 필요합니다.
- **actions** 매개 변수는 구성을 적용할 시기를 정의합니다. 이 경우 오버클라우드를 생성하

거나 업데이트하는 경우에만 구성을 적용합니다. 가능한 작업에는 **CREATE,UPDATE,DELETE,SUSPEND, RESUME** 등이 있습니다.

- **input_values** 매개 변수에는 상위 템플릿에서 **DeployIdentifier** 를 저장하는 **deploy_identifier** 라는 하위 매개 변수가 포함되어 있습니다. 이 매개 변수는 각 배포 업데이트의 리소스에 타임스탬프를 제공합니다. 이렇게 하면 리소스가 후속 오버클라우드 업데이트에 다시 적용됩니다.

다음으로 **heat** 템플릿을 **OS::TripleO::NodeExtraConfig** 리소스 유형으로 등록하는 환경 파일 (**/home/stack/templates/pre_config.yaml**)을 만듭니다.

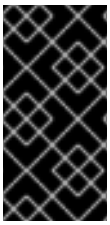
```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

구성을 적용하려면 **Overcloud**를 생성하거나 업데이트할 때 다른 환경 파일과 함께 스택에 환경 파일을 추가합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

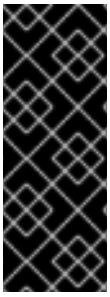
이는 초기 **Overcloud** 생성 또는 후속 업데이트에서 코어 구성이 시작되기 전에 모든 노드에 구성이 적용됩니다.



중요

OS::TripleO::NodeExtraConfig 를 하나의 **Heat** 템플릿에만 등록할 수 있습니다. 후속 사용량은 사용할 **Heat** 템플릿을 재정의합니다.

4.4. 구성 후: 모든 오버클라우드 역할 사용자 정의



중요

이 문서의 이전 버전에서는 **OS::TripleO::Tasks::*PostConfig** 리소스를 사용하여 역할별로 사후 구성 후크를 제공했습니다. **director**의 **Heat** 템플릿 컬렉션에는 이러한 후크를 전용으로 사용해야 하므로 사용자 지정 용도로 사용해서는 안 됩니다. 대신 아래에 설명된 **OS::TripleO::NodeExtraConfigPost** 후크를 사용합니다.

오버클라우드 생성을 완료했지만 초기 생성 또는 오버클라우드의 후속 업데이트 시 모든 역할에 구성을 추가하려는 경우 발생할 수 있습니다. 이 경우 다음 구성 후 후크를 사용합니다.

OS::TripleO::NodeExtraConfigPost

핵심 **Puppet** 구성 후 모든 노드 역할에 적용되는 추가 구성입니다.

이 예제에서는 먼저 스크립트를 실행하여 각 노드의 **resolv.conf**를 변수 이름 서버로 추가하는 기본 **heat** 템플릿(**/home/stack/templates/nameserver.yaml**)을 생성합니다.

```
description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeployments:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      servers:  {get_param: servers}
      config: {get_resource: CustomExtraConfig}
```

```
    actions: ['CREATE','UPDATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}
```

이 예제에서 **resources** 섹션에는 다음이 포함됩니다.

CustomExtraConfig

소프트웨어 구성을 정의합니다. 이 예제에서는 **Bash** 스크립트를 정의하고 **Heat**에서 **_NAMESERVER_IP_** 를 **nameserver_ip** 매개 변수에 저장된 값으로 바꿉니다.

CustomExtraDeployments

그러면 **CustomExtraConfig** 리소스의 소프트웨어 구성인 소프트웨어 구성이 실행됩니다. 다음을 확인합니다.

- **config** 매개변수는 **CustomExtraConfig** 리소스에 대한 참조를 만들어 **Heat**에서 적용할 구성을 알 수 있습니다.
- **servers** 매개 변수는 **Overcloud** 노드 맵을 검색합니다. 이 매개 변수는 상위 템플릿에서 제공하며 이 후크의 템플릿에서 필요합니다.
- **actions** 매개 변수는 구성을 적용할 시기를 정의합니다. 이 경우 **Overcloud**가 생성될 때만 구성을 적용합니다. 가능한 작업에는 **CREATE,UPDATE,DELETE,SUSPEND, RESUME** 등이 있습니다.
- **input_values** 에는 상위 템플릿에서 **DeployIdentifier** 를 저장하는 **deploy_identifier** 라는 매개 변수가 포함되어 있습니다. 이 매개 변수는 각 배포 업데이트의 리소스에 타임스탬프를 제공합니다. 이렇게 하면 리소스가 후속 오버클라우드 업데이트에 다시 적용됩니다.

다음으로 **heat** 템플릿을 **OS::TripleO::NodeExtraConfigPost:** 리소스 유형으로 등록하는 환경 파일 (**/home/stack/templates/post_config.yaml**)을 생성합니다.

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

구성을 적용하려면 **Overcloud**를 생성하거나 업데이트할 때 다른 환경 파일과 함께 스택에 환경 파일을

추가합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  ...
  -e /home/stack/templates/post_config.yaml \
  ...
```

이는 초기 **Overcloud** 생성 또는 후속 업데이트에서 코어 구성이 완료된 후 모든 노드에 구성이 적용됩니다.

중요

OS::TripleO::NodeExtraConfigPost 를 하나의 **Heat** 템플릿에만 등록할 수 있습니다. 후속 사용량은 사용할 **Heat** 템플릿을 재정의합니다.

4.5. PUPPET: 역할에 맞는 HIERADATA 사용자 정의

Heat 템플릿 컬렉션에는 특정 노드 유형에 추가 구성을 전달하는 매개변수 세트가 포함되어 있습니다. 이러한 매개 변수는 노드의 **Puppet** 구성에 대한 **hieradata**로 구성을 저장합니다. 이러한 매개변수는 다음과 같습니다.

ControllerExtraConfig

모든 컨트롤러 노드에 추가할 구성입니다.

ComputeExtraConfig

모든 컴퓨팅 노드에 추가할 구성입니다.

BlockStorageExtraConfig

모든 블록 스토리지 노드에 추가할 구성입니다.

ObjectStorageExtraConfig

모든 **Object Storage** 노드에 추가할 구성

CephStorageExtraConfig

모든 **Ceph Storage** 노드에 추가할 구성

[ROLE]ExtraConfig

구성 가능 역할에 추가할 구성입니다. **[ROLE]** 을 구성 가능 역할 이름으로 교체합니다.

ExtraConfig

모든 노드에 추가할 구성입니다.

배포 후 구성 프로세스에 구성을 추가하려면 **parameter_defaults** 섹션에 이러한 매개 변수를 포함하는 환경 파일을 생성합니다. 예를 들어 **Compute** 호스트의 예약된 메모리를 **1024MB**로 늘리고 **VNC** 키맵을 **Japanese**로 설정하려면 다음을 실행합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
    nova::compute::vnc_keymap: ja
```

openstack overcloud deploy 를 실행할 때 이 환경 파일을 포함합니다.

중요

각 매개 변수는 한 번만 정의할 수 있습니다. 후속 사용은 이전 값을 재정의합니다.

4.6. PUPPET: 개별 노드에 대한 HIERADATA 사용자 정의

Heat 템플릿 컬렉션을 사용하여 개별 노드에 **Puppet hieradata**를 설정할 수 있습니다. 이를 수행하려면 노드의 인트로스펙션 데이터의 일부로 시스템 **UUID**를 저장해야 합니다.

```
$ openstack baremetal introspection data save 9dcc87ae-4c6d-4ede-81a5-9b20d7dc4a14 | jq
.extra.system.product.uuid
```

이렇게 하면 시스템 **UUID**가 출력됩니다. 예를 들면 다음과 같습니다.

```
"F5055C6C-477F-47FB-AFE5-95C6928C407F"
```

노드별 **hieradata**를 정의하고 **per_node.yaml** 템플릿을 사전 구성 후크에 등록하는 환경 파일에서 이 시스템 **UUID**를 사용합니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::ComputeExtraConfigPre: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/pre_deploy/per_node.yaml
```

```
parameter_defaults:
  NodeDataLookup: '{"F5055C6C-477F-47FB-AFE5-95C6928C407F":
{"nova::compute::vcpu_pin_set": [ "2", "3" ]}}'
```

openstack overcloud deploy 를 실행할 때 이 환경 파일을 포함합니다.

per_node.yaml 템플릿은 각 시스템 **UUID**에 해당하는 노드에 **heiradata** 파일 세트를 생성하고 사용자가 정의한 **hieradata**를 포함합니다. **UUID**가 정의되지 않은 경우 결과 **hieradata** 파일이 비어 있습니다. 이전 예에서 **per_node.yaml** 템플릿은 **OS::TripleO::ComputeExtraConfigPre** 후크에 따라 모든 컴퓨팅 노드에서 실행되지만 시스템 **UUID F5055C6C-477F-47FB-AFE5-95C6928C407F** 가 있는 컴퓨팅 노드만 **hieradata**를 수신합니다.

이렇게 하면 각 노드를 특정 요구 사항에 맞게 조정할 수 있습니다.

NodeDataLookup에 대한 자세한 내용은 ***Deploy ing an Overcloud with Containerized Red Hat Ceph 가이드의 Ceph Storage Cluster Setting*** 구성을 참조하십시오.

4.7. PUPPET: 사용자 정의 매니페스트 적용

특정 상황에서는 오버클라우드 노드에 추가 구성 요소를 설치하고 구성해야 할 수 있습니다. 기본 구성이 완료된 후 의 노드에 적용되는 사용자 지정 **Puppet** 매니페스트를 사용하여 이 작업을 수행할 수 있습니다. 기본 예제에서는 **motd** 를 각 노드에 설치할 수 있습니다. 이 작업을 수행하는 프로세스는 먼저 **Puppet** 구성을 시작하는 **Heat** 템플릿(**/home/stack/templates/custom_puppet_config.yaml**)을 생성하는 것입니다.

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json

resources:
  ExtraPuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      config: {get_file: motd.pp}
      group: puppet
      options:
        enable_hiera: True
        enable_facter: False

  ExtraPuppetDeployments:
```

```
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      config: {get_resource: ExtraPuppetConfig}
      servers: {get_param: servers}
```

여기에는 템플릿 내에 **/home/stack/templates/motd.pp** 이 포함되어 구성을 위해 노드에 전달합니다. **motd.pp** 파일 자체에는 **motd**를 설치하고 구성하는 **Puppet** 클래스가 포함되어 있습니다 **.**

다음으로 **heat** 템플릿을 **OS::TripleO::NodeExtraConfigPost:** 리소스 유형으로 등록하는 환경 파일 **(/home/stack/templates/puppet_post_config.yaml)**을 생성합니다.

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/custom_puppet_config.yaml
```

마지막으로 오버클라우드 스택을 생성하거나 업데이트할 때 다른 환경 파일과 함께 이 환경 파일을 추가합니다.

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/puppet_post_config.yaml \
    ...
```

그러면 **motd.pp** 의 구성이 **Overcloud**의 모든 노드에 적용됩니다.

중요

동일한 사용자 지정 **hieradata** 해시의 여러 인스턴스를 정의하지 마십시오. 동일한 사용자 지정 **hieradata**의 여러 인스턴스로 인해 **Puppet** 실행 중에 충돌하여 구성 옵션에 대해 예기치 않은 값이 설정될 수 있습니다.

5장. 오버클라우드 등록

오버클라우드는 **Red Hat Content Delivery Network, Red Hat Satellite Server 5** 또는 **Red Hat Satellite Server 6**에 노드를 등록할 수 있는 방법을 제공합니다.

5.1. 환경 파일로 오버클라우드 등록

Heat 템플릿 컬렉션에서 등록 파일을 복사합니다.

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration
~/templates/.
```

~/templates/rhel-registration/environment-rhel-registration.yaml 파일을 편집하고 등록 메서드 및 세부 정보에 적용되는 매개변수 값을 변경합니다.

일반 매개변수

rhel_reg_method

등록 방법을 선택합니다. **portal,satellite** 또는 **disable** 중 하나입니다.

rhel_reg_type

등록할 유닛 유형입니다. 시스템으로등록하려면 비워 두십시오

rhel_reg_auto_attach

이 시스템에 호환 가능한 서브스크립션을 자동으로 첨부합니다. 활성화하려면 **true** 로 설정합니다. 이 기능을 비활성화하려면 이 매개 변수를 비워 둡니다.

rhel_reg_service_level

자동 첨부에 사용할 서비스 수준입니다.

rhel_reg_release

자동 첨부를 위한 릴리스 버전을 설정하려면 이 매개변수를 사용합니다. **Red Hat** 서브스크립션 관리자의 기본값을 사용하려면 비워 두십시오.

rhel_reg_pool_id

사용하려는 서브스크립션 풀 **ID**입니다. 서브스크립션을 자동 첨부하지 않는 경우 이 옵션을 사용합니다. 이 **ID**를 찾으려면 언더클라우드 노드에서 **sudo subscription-manager list --available --all -**

-matches="*OpenStack*" 을 실행하고 결과 풀 ID 값을 사용합니다.

rhel_reg_sat_url

오버클라우드 노드를 등록할 Satellite Server의 기본 URL입니다. Satellite Server HTTP URL 대신 이 매개 변수의 HTTPS URL을 사용합니다. 예를 들어 https://satellite.example.com 대신 http://satellite.example.com을 사용합니다. 오버클라우드 생성 프로세스에서는 이 URL을 사용하여 Red Hat Satellite Server 5 또는 Red Hat Satellite Server 6을 사용 중인지 확인합니다. Red Hat Satellite Server 6을 사용하는 경우 오버클라우드는 katello-ca-consumer-latest.noarch.rpm 파일을 가져오고 subscription-manager 에 등록하고 katello-agent 를 설치합니다. Red Hat Satellite Server 5를 사용하는 경우 오버클라우드는 RHN-ORG-TRUSTED-SSL-CERT 파일을 가져오고 rhnreg_ks 에 등록합니다.

rhel_reg_server_url

사용하려는 서브스크립션 서비스의 호스트 이름입니다. 고객 포털 서브스크립션 관리의 기본값은 subscription.rhn.redhat.com입니다. 이 옵션을 사용하지 않는 경우 시스템은 고객 포털 서브스크립션 관리에 등록됩니다. 서브스크립션 서버 URL은 https://hostname:port/prefix 형식을 사용합니다.

rhel_reg_base_url

업데이트를 수신하는 데 사용할 콘텐츠 전달 서버의 호스트 이름입니다. 기본값은 https://cdn.redhat.com 입니다. Satellite 6은 자체 콘텐츠를 호스팅하므로 Satellite 6에 등록된 시스템에 URL을 사용해야 합니다. 콘텐츠의 기본 URL에서는 https://hostname:port/prefix 형식을 사용합니다.

rhel_reg_org

등록에 사용하려는 조직입니다. 이 ID를 찾으려면 언더클라우드 노드에서 sudo subscription-manager orgs 를 실행합니다. 메시지가 표시되면 Red Hat 자격 증명을 입력하고 결과 키 값을 사용합니다.

rhel_reg_environment

선택한 조직 내에서 사용하려는 환경입니다.

rhel_reg_repos

활성화할 쉼표로 구분된 리포지토리 목록입니다.

rhel_reg_activation_key

등록에 사용할 활성화 키입니다. 등록에 활성화 키를 사용하는 경우 등록에 사용할 조직도 지정해야 합니다.

rhel_reg_user; rhel_reg_password

등록에 사용할 사용자 이름과 암호입니다. 가능한 경우 등록에 활성화 키를 사용합니다.

rhel_reg_machine_name

등록에 사용할 시스템 이름입니다. 노드의 호스트 이름을 사용하려면 이 비워 두십시오.

rhel_reg_force

노드를 다시 등록할 때 등록 옵션을 강제 적용하려면 **true** 로 설정합니다.

rhel_reg_sat_repo

Red Hat Satellite 6 서버 관리 툴(예: **katello-agent**)이 포함된 리포지토리입니다. 리포지토리 이름이 **Satellite Server** 버전에 해당하고 리포지토리가 **Satellite Server**에 동기화되었는지 확인합니다. 예를 들어 **rhel-7-server-satellite-tools-6.2-rpms** 는 **Red Hat Satellite 6.2**에 해당합니다.

업그레이드 매개 변수

UpdateOnRHELRegistration

True 로 설정하면 등록이 완료되면 오버클라우드 패키지 업데이트가 트리거됩니다. 기본적으로 **False**로 설정합니다.

HTTP 프록시 매개 변수

rhel_reg_http_proxy_host

HTTP 프록시의 호스트 이름입니다. 예: **proxy.example.com**

rhel_reg_http_proxy_port

HTTP 프록시 통신용 포트입니다. 예를 들면 다음과 같습니다. **8080**.

rhel_reg_http_proxy_username

HTTP 프록시에 액세스할 사용자 이름입니다.

rhel_reg_http_proxy_password

HTTP 프록시에 액세스할 암호입니다.

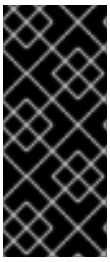
중요

프록시 서버를 사용하는 경우 모든 오버클라우드 노드에 **rhel_reg_http_proxy_host** 매개 변수에 정의된 호스트 경로가 있는지 확인합니다. 이 호스트의 경로가 없으면 **subscription-manager** 가 시간 초과되고 배포 실패가 발생합니다.

배포 명령(**openstack overcloud deploy**)은 **-e** 옵션을 사용하여 환경 파일을 추가합니다.

~/templates/rhel-registration/environment-rhel-registration.yaml 및 **~/templates/rhel-registration/rhel-registration-resource-registry.yaml** 을 둘 다 추가합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates [...] -e /home/stack/templates/rhel-registration/environment-rhel-registration.yaml -e /home/stack/templates/rhel-registration/rhel-registration-resource-registry.yaml
```



중요

등록은 **OS::TripleO::NodeExtraConfig Heat** 리소스로 설정됩니다. 즉, 등록에는 이 리소스만 사용할 수 있습니다. 자세한 내용은 **4.2**절. "구성 전: 특정 오버클라우드 역할 사용자 정의"를 참조하십시오.

5.2. 예 1: 고객 포털에 등록

다음은 **my-openstack** 활성화 키를 사용하여 오버클라우드 노드를 **Red Hat** 고객 포털에 등록하고 **1a85f9223e3d5e43013e3e8ff506fd** 풀을 서브스크립션합니다.

```
parameter_defaults:
  rhel_reg_auto_attach: ""
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1234567"
  rhel_reg_pool_id: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhel_reg_repos: "rhel-7-server-rpms,rhel-7-server-extras-rpms,rhel-7-server-rh-common-rpms,rhel-ha-for-rhel-7-server-rpms,rhel-7-server-openstack-13-rpms,rhel-7-server-rhceph-3-osd-rpms,rhel-7-server-rhceph-3-mon-rpms,rhel-7-server-rhceph-3-tools-rpms"
  rhel_reg_method: "portal"
  rhel_reg_sat_repo: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_release: ""
  rhel_reg_sat_url: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

5.3. 예 2: RED HAT SATELLITE 6 SERVER에 등록

다음은 **sat6.example.com**의 **Red Hat Satellite 6 Server**에 오버클라우드 노드를 등록하고 **my-**

openstack 활성화 키를 사용하여 풀 1a85f9223e3d5e43013e3d6e8ff506fd 에 가입합니다. 이 경우 활성화 키는 활성화할 리포지토리도 제공합니다.

```
parameter_defaults:
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1"
  rhel_reg_pool_id: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhel_reg_method: "satellite"
  rhel_reg_sat_url: "http://sat6.example.com"
  rhel_reg_sat_repo: "rhel-7-server-satellite-tools-6.2-rpms"
  rhel_reg_repos: ""
  rhel_reg_auto_attach: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_release: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

5.4. 예 3: RED HAT SATELLITE 5 SERVER에 등록

다음은 sat5.example.com의 Red Hat Satellite 5 Server에 오버클라우드 노드를 등록하고 my-openstack 활성화 키를 사용하여 서브스크립션을 자동으로 연결합니다. 이 경우 활성화 키는 활성화할 리포지토리도 제공합니다.

```
parameter_defaults:
  rhel_reg_auto_attach: ""
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1"
  rhel_reg_method: "satellite"
  rhel_reg_sat_url: "http://sat5.example.com"
  rhel_reg_repos: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_pool_id: ""
  rhel_reg_release: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
```

```
  rhel_reg_sat_repo: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

5.5. 예 4: HTTP 프록시를 통한 등록

다음 샘플 매개변수는 원하는 등록 방법에 대한 **HTTP** 프록시 설정을 설정합니다.

```
parameter_defaults:
  ...
  rhel_reg_http_proxy_host: "proxy.example.com"
  rhel_reg_http_proxy_port: "8080"
  rhel_reg_http_proxy_username: "proxyuser"
  rhel_reg_http_proxy_password: "p@55w0rd!"
  ...
```

5.6. 고급 등록 방법

경우에 따라 다른 서브스크립션 유형에 다른 역할을 등록해야 할 수 있습니다. 예를 들어 컨트롤러 노드만 **OpenStack Platform** 서브스크립션에 서브스크립션하고 **Ceph Storage** 노드를 **Ceph Storage** 서브스크립션에 가입하려고 할 수 있습니다. 이 섹션에서는 다양한 역할에 별도의 서브스크립션을 할당하는 데 도움이 되는 몇 가지 고급 등록 방법을 제공합니다.

구성 후크

한 가지 방법은 역할별 스크립트를 작성하고 역할별 후크를 사용하여 포함하는 것입니다. 예를 들어 다음 코드 조각을 **OS::TripleO::ControllerExtraConfigPre** 리소스의 템플릿에 추가하여 컨트롤러 노드만 이러한 서브스크립션 세부 정보를 수신할 수 있습니다.

```
ControllerRegistrationConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: script
    config: |
      #!/bin/sh
      sudo subscription-manager register --org 1234567 \
        --activationkey "my-openstack"
      sudo subscription-manager attach --pool 1a85f9223e3d5e43013e3d6e8ff506fd
      sudo subscription-manager repos --enable rhel-7-server-rpms \
        --enable rhel-7-server-extras-rpms \
        --enable rhel-7-server-rh-common-rpms \
        --enable rhel-ha-for-rhel-7-server-rpms \
        --enable rhel-7-server-openstack-13-rpms \
        --enable rhel-7-server-rhceph-3-mon-rpms

ControllerRegistrationDeployment:
```

```
  type: OS::Heat::SoftwareDeployment
  properties:
    server: {get_param: server}
    config: {get_resource: ControllerRegistrationConfig}
    actions: ['CREATE','UPDATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}
```

이 스크립트는 **subscription-manager** 명령 집합을 사용하여 시스템을 등록하고, 서브스크립션을 연결하며, 필요한 리포지토리를 활성화합니다.

후크에 대한 자세한 내용은 **4**장. *구성 후크* 을 참조하십시오.

Ansible 기반 구성

director의 동적 인벤토리 스크립트를 사용하여 특정 역할에 대한 **Ansible** 기반 등록을 수행할 수 있습니다. 예를 들어 다음 플레이를 사용하여 컨트롤러 노드를 등록하려고 할 수 있습니다.

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-mon-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        activationkey: my-openstack
        org_id: 1234567
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

이 플레이에는 세 가지 작업이 포함되어 있습니다. **-** 활성화 키를 사용하여 노드 등록 **-** 자동 사용 리포지토리 비활성화 **-** 컨트롤러 노드와 관련된 리포지토리만 활성화합니다. 리포지토리는 **repos** 변수를 사용하여 나열됩니다.

오버클라우드를 배포한 후 **Ansible**이 오버클라우드에 대해 플레이북**(ansible-osp-registration.yml)**

을 실행하도록 다음 명령을 실행할 수 있습니다.

```
$ ansible-playbook -i /usr/bin/tripleo-ansible-inventory ansible-osp-registration.yml
```

이 명령은 다음을 수행합니다. - 동적 인벤토리 스크립트를 실행하여 호스트 및 그룹 목록을 가져옵니다. - 플레이북의 **hosts** 매개 변수에 정의된 그룹의 노드에 플레이북 작업을 적용합니다. 이 경우에는 **Controller** 그룹입니다.

오버클라우드에서 실행 중인 **Ansible** 자동화에 대한 자세한 내용은 ***Director** 설치 및 사용* 가이드의 **"Ansible** 자동화 실행**"** 을 참조하십시오.

6장. ANSIBLE 기반 오버클라우드 등록

중요

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

5장. 오버클라우드 등록 의 **rhel-registration** 방법에 대한 대안으로 **director**는 **Ansible** 기반 방법을 사용하여 오버클라우드 노드를 **Red Hat Customer Portal** 또는 **Red Hat Satellite 6** 서버에 등록할 수 있습니다. 이 방법은 오버클라우드에서 **Ansible** 기반구성(**config-download**)을 활성화하는 데 사용합니다.

6.1. RHSM(RED HAT SUBSCRIPTION MANAGER) 구성 가능 서비스

The rhsm 구성 가능 서비스는 **Ansible**을 통해 오버클라우드 노드를 등록하는 방법을 제공합니다. 기본 **roles_data** 파일의 각 역할에는 기본적으로 비활성화된 **OS::TripleO::Services::Rhsm** 리소스가 포함되어 있습니다. 서비스를 활성화하려면 리소스를 **rhsm** 구성 가능 서비스 파일에 등록합니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
```

The rhsm 구성 가능 서비스는 **RhsmVars** 매개변수를 허용하므로 등록과 관련된 여러 하위 매개변수를 정의할 수 있습니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-osd-rpms
      - rhel-7-server-rhceph-3-mon-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    rhsm_org_id: "1234567"
```

역할별 매개변수(예: **ControllerParameter s**)와 함께 **RhsmVar s**매개변수를 사용하여 다른 노드 유형에 특정 리포지토리를 활성화할 때 유연성을 제공할 수도 있습니다.

다음 섹션은 Rhsm 구성 가능 서비스와 함께 사용할 수 있는 하위 매개변수 목록입니다.

6.2. RHSMVARS 하위 매개변수

rhsm	설명
rhsm_method	등록 방법을 선택합니다. **portal**,**satellite** 또는 **disable** 중 하나입니다.
rhsm_org_id	등록에 사용할 조직입니다. 이 ID를 찾으려면 언더클라우드 노드에서 **sudo subscription-manager orgs** 를 실행합니다. 메시지가 표시되면 Red Hat 자격 증명을 입력하고 결과 **키** 값을 사용합니다.
rhsm_pool_ids	사용할 서브스크립션 풀 ID입니다. 서브스크립션을 자동 첨부하지 않는 경우 이 옵션을 사용합니다. 이 ID를 찾으려면 언더클라우드 노드에서 **sudo subscription-manager list --available --all --matches="*OpenStack*"** 을 실행하고 결과 **풀 ID** 값을 사용합니다.
rhsm_activation_key	등록에 사용할 활성화 키입니다.
rhsm_autosubscribe	이 시스템에 호환 가능한 서브스크립션을 자동으로 첨부합니다. 활성화하려면 **true** 로 설정합니다.
rhsm_satellite_url	오버클라우드 노드를 등록할 Satellite 서버의 기본 URL입니다.
rhsm_repos	활성화할 리포지토리 목록입니다.
rhsm_username	등록할 사용자 이름입니다. 가능한 경우 등록에 활성화 키를 사용합니다.
rhsm_password	등록할 암호입니다. 가능한 경우 등록에 활성화 키를 사용합니다.
rhsm_rhsm_proxy_hostname	HTTP 프록시의 호스트 이름입니다. 예: **proxy.example.com**
rhsm_rhsm_proxy_port	HTTP 프록시 통신용 포트입니다. 예를 들면 다음과 같습니다. **8080**.
rhsm_rhsm_proxy_user	HTTP 프록시에 액세스할 사용자 이름입니다.
rhsm_rhsm_proxy_password	HTTP 프록시에 액세스할 암호입니다.

이제 the rhsm 구성 가능 서비스 작동 방식과 구성 방법에 대해 이해했으므로 다음 절차를 사용하여 고유한 등록 세부 정보를 구성할 수 있습니다.

6.3. RHSM 구성 가능 서비스를 사용하여 오버클라우드 등록

다음 절차에 따라 구성 가능 서비스를 활성화하고 구성하는 환경 파일을 생성합니다. **director**는 이 환경 파일을 사용하여 노드를 등록하고 서브스크립션합니다.

절차

1. 환경 파일(**templates/rhsm.yml**)을 만들어 구성을 저장합니다.

2. 환경 파일에 구성을 포함합니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-osd-rpms
      - rhel-7-server-rhceph-3-mon-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
```

resource_registry 는 각 역할에서 사용할 수 있는 **OS::TripleO::Services::Rhsm** 리소스와 **rhsm** 구성 가능 서비스를 연결합니다.

RhsmVars 변수는 **Red Hat** 등록을 구성하기 위해 매개 변수를 **Ansible**에 전달합니다.

3. 환경 파일 저장

특정 오버클라우드 역할에 대한 등록 세부 정보도 제공할 수 있습니다. 다음 섹션에서는 이 예제를 제공합니다.

6.4. 다른 역할에 RHSM 구성 가능 서비스 적용

역할별로 **the rhsm** 구성 가능 서비스를 적용할 수 있습니다. 예를 들어 컨트롤러 노드에 하나의 구성 세트를 적용하고 다른 구성 세트를 컴퓨팅 노드에 적용할 수 있습니다.

절차

1. 환경 파일**(templates/rhsm.yml)**을 만들어 구성을 저장합니다.

2. 환경 파일에 구성을 포함합니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-7-server-rpms
        - rhel-7-server-extras-rpms
        - rhel-7-server-rh-common-rpms
        - rhel-ha-for-rhel-7-server-rpms
        - rhel-7-server-openstack-13-rpms
        - rhel-7-server-rhceph-3-osd-rpms
        - rhel-7-server-rhceph-3-mon-rpms
        - rhel-7-server-rhceph-3-tools-rpms
      rhsm_activation_key: "my-openstack"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
      rhsm_method: "portal"
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-7-server-rpms
        - rhel-7-server-extras-rpms
        - rhel-7-server-rh-common-rpms
        - rhel-ha-for-rhel-7-server-rpms
        - rhel-7-server-openstack-13-rpms
        - rhel-7-server-rhceph-3-tools-rpms
      rhsm_activation_key: "my-openstack"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
      rhsm_method: "portal"
```

resource_registry 는 각 역할에서 사용할 수 있는 **OS::TripleO::Services::Rhsm** 리소스와 **rhsm** 구성 가능 서비스를 연결합니다.

ControllerParameters 및 **ComputeParameters** 둘 다 고유한 **RhsmVars** 매개변수를 사용하여 서브스크립션 세부 정보를 해당 역할에 전달합니다.

3. 환경 파일 저장

6.5. RED HAT SATELLITE SERVER에 오버클라우드 등록

Red Hat 고객 포털 대신 Red Hat Satellite에 노드를 등록하도록 rhsm 구성 가능 서비스를 활성화하고 구성하는 환경 파일을 생성합니다.

절차

1. templates/rhsm.yml 이라는 환경 파일을 생성하여 구성을 저장합니다.

2. 환경 파일에 구성을 포함합니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
parameter_defaults:
  RhsmVars:
    rhsm_activation_key: "myactivationkey"
    rhsm_method: "satellite"
    rhsm_org_id: "ACME"
    rhsm_server_hostname: "satellite.example.com"
    rhsm_baseurl: "https://satellite.example.com/pulp/repos"
    rhsm_release: 7.9
```

resource_registry 는 각 역할에서 사용할 수 있는 OS::TripleO::Services::Rhsm 리소스와 rhsm 구성 가능 서비스를 연결합니다.

RhsmVars 변수는 Red Hat 등록을 구성하기 위해 매개 변수를 Ansible에 전달합니다.

3. 환경 파일을 저장합니다.

다음 절차에서는 오버클라우드 에서rhsm 을 활성화하고 구성합니다. 그러나 5장. *오버클라우드 등록* 의 rhel-registration 방법을 사용하는 경우 Ansible 기반 방법으로 전환하려면 비활성화해야 합니다. rhel-registration 방법에서 Ansible 기반 방법으로 전환하려면 다음 절차를 사용합니다.

6.6. RHSM 구성 가능 서비스로 전환

Overcloud 등록을 처리하는 **bash** 스크립트를 실행하는 **rhel-registration** 메서드입니다. 이 메서드의 스크립트 및 환경 파일은 **/usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/** 의 코어 **Heat** 템플릿 컬렉션에 있습니다.

다음 절차에서는 **rhel-registration** 방법에서 구성 가능 서비스로 전환하는 방법을 설명합니다.

절차

1. **rhel-registration** 환경 파일을 향후 배포 작업에서 제외합니다. 대부분의 경우 다음과 같은 파일이 됩니다.
   - **rhel-registration/environment-rhel-registration.yaml**
   - **rhel-registration/rhel-registration-resource-registry.yaml**
2. 향후 배포 작업에 환경 파일 **for rhsm composable** 서비스 매개변수를 추가합니다.

이 메서드는 **rhel-registration** 매개변수를 **rhsm** 서비스 매개변수로 교체하고 다음과 같이 서비스를 활성화하는 **Heat** 리소스를 변경합니다.

```
resource_registry:
  OS::TripleO::NodeExtraConfig: rhel-registration.yaml
```

다음으로 변경합니다.

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
```

rhel-registration 메서드에서 **rhsm** 메서드로 세부 정보를 전환하는 데 도움이 되도록 다음 표를 사용하여 매개변수 및 해당 값을 매핑합니다.

6.7. RHEL-REGISTRATION TO RHSM MAPPINGS

rhel-registration	rhsm / RhsmVars
rhel_reg_method	rhsm_method
rhel_reg_org	rhsm_org_id
rhel_reg_pool_id	rhsm_pool_ids
rhel_reg_activation_key	rhsm_activation_key
rhel_reg_auto_attach	rhsm_autosubscribe
rhel_reg_sat_url	rhsm_satellite_url
rhel_reg_repos	rhsm_repos
rhel_reg_user	rhsm_username
rhel_reg_password	rhsm_password
rhel_reg_http_proxy_host	rhsm_rhsm_proxy_hostname
rhel_reg_http_proxy_port	rhsm_rhsm_proxy_port
rhel_reg_http_proxy_username	rhsm_rhsm_proxy_user
rhel_reg_http_proxy_password	rhsm_rhsm_proxy_password

이제 **rhsm** 서비스에 대한 환경 파일을 구성했으므로 다음 오버클라우드 배포 작업을 통해 이 파일을 포함할 수 있습니다.

6.8. RHSM 구성 가능 서비스를 사용하여 오버클라우드 배포

다음 프로세스에서는 오버클라우드에 **your rhsm** 구성을 적용하는 방법을 보여줍니다.

절차

1. **openstack overcloud deploy** 명령을 실행하는 경우 **config-download** 옵션 및 환경 파일과 **rhsm.yml** 환경 파일을 포함합니다.

```
openstack overcloud deploy \
  <other cli args> \
```

```
    -e /usr/share/openstack-tripleo-heat-templates/environments/config-download-
environment.yaml \
    --config-download \
    -e ~/templates/rhsm.yaml
```

이렇게 하면 오버클라우드의 **Ansible** 구성 및 **Ansible** 기반 등록이 활성화됩니다.

2. **Overcloud** 배포가 완료될 때까지 기다립니다.

3. 오버클라우드 노드에서 서브스크립션 세부 정보를 확인합니다. 예를 들어 컨트롤러 노드에 로그인하고 다음 명령을 실행합니다.

```
$ sudo subscription-manager status
$ sudo subscription-manager list --consumed
```

7장. 구성 가능 서비스 및 사용자 지정 역할

Overcloud는 일반적으로 컨트롤러 노드, 컴퓨팅 노드 및 다양한 스토리지 노드 유형과 같은 사전 정의된 역할의 노드로 구성됩니다. 이러한 각 기본 역할에는 director 노드의 코어 Heat 템플릿 컬렉션에 정의된 서비스 세트가 포함되어 있습니다. 그러나 핵심 Heat 템플릿의 아키텍처에서는 다음을 수행하는 방법을 제공합니다.

- 사용자 지정 역할 만들기
- 각 역할에서 서비스 추가 및 제거

이를 통해 다양한 역할에 서비스 조합을 생성할 수 있습니다. 이 장에서는 사용자 지정 역할, 구성 가능 서비스 및 이를 사용하는 방법에 대해 살펴봅니다.

7.1. 지원되는 역할 아키텍처

사용자 지정 역할 및 구성 가능한 서비스를 사용할 때 다음과 같은 아키텍처를 사용할 수 있습니다.

아키텍처 1 - 기본 아키텍처

기본 roles_data 파일을 사용합니다. 모든 컨트롤러 서비스는 하나의 컨트롤러 역할 내에 포함됩니다.

아키텍처 2 - 지원되는 독립 실행형 역할

/usr/share/openstack-tripleo-heat-templates/roles 에서 사전 정의된 파일을 사용하여 사용자 지정 roles_data 파일'을 생성합니다. 7.2.3절. “지원되는 사용자 지정 역할”을 참조하십시오.

아키텍처 3 - 맞춤형 구성 가능 서비스

고유한 역할을 생성하고 이를 사용하여 사용자 지정 roles_data 파일을 생성합니다. 제한된 수의 구성 가능 서비스 조합만 테스트 및 검증되었으며 Red Hat은 모든 구성 가능한 서비스 조합을 지원할 수 없습니다.

7.2. 역할

7.2.1. roles_data 파일 검사

Overcloud 생성 프로세스는 roles_data 파일을 사용하여 역할을 정의합니다. roles_data 파일에는 YAML 형식의 역할 목록이 포함되어 있습니다. 다음은 roles_data 구문의 단축된 예입니다.

```
- name: Controller
  description: |
    Controller role that has all the controler services loaded and handles
    Database, Messaging and Network functions.
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  description: |
    Basic Compute Node role
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

코어 **Heat** 템플릿 컬렉션에는 **/usr/share/openstack-tripleo-heat-templates/roles_data.yaml**에 있는 기본 **roles_data** 파일이 포함되어 있습니다. 기본 파일은 다음 역할 유형을 정의합니다.

- 컨트롤러
- **Compute**
- **BlockStorage**
- **ObjectStorage**
- **CephStorage.**

openstack overcloud deploy 명령에는 배포 중에 이 파일이 포함됩니다. **r** 인수를 사용하여 이 파일을 사용자 지정 **roles_data** 파일로 재정의할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-custom.yaml
```

7.2.2. roles_data 파일 생성

사용자 지정 **roles_data** 파일을 수동으로 생성할 수 있지만 개별 역할 템플릿을 사용하여 파일을 자동으로 생성할 수도 있습니다. **director**는 역할 템플릿을 관리하고 사용자 지정 **roles_data** 파일을 자동으로 생성하는 여러 명령을 제공합니다.

기본 역할 템플릿을 나열하려면 **openstack overcloud role list** 명령을 사용합니다.

```
$ openstack overcloud role list
BlockStorage
CephStorage
Compute
ComputeHCI
ComputeOvsDpdk
Controller
...
```

역할의 **YAML** 정의를 보려면 **openstack overcloud role show** 명령을 사용합니다.

```
$ openstack overcloud role show Compute
```

사용자 지정 **roles_data** 파일을 생성하려면 **openstack overcloud roles generate** 명령을 사용하여 사전 정의된 여러 역할을 하나의 파일에 결합합니다. 예를 들어 다음 명령은 **Controller,Compute** 및 **Networker** 역할을 단일 파일에 결합합니다.

```
$ openstack overcloud roles generate -o ~/roles_data.yaml Controller Compute Networker
```

o는 생성할 파일의 이름을 정의합니다.

이렇게 하면 사용자 지정 **roles_data** 파일이 생성됩니다. 그러나 이전 예제에서는 모두 동일한 네트워킹 에이전트를 포함하는 **Controller** 및 **Networker** 역할을 사용합니다. 즉, 네트워킹 서비스가 컨트롤러에서 **Networker** 역할로 확장됩니다. **Overcloud**에서 컨트롤러 노드와 네트워크 노드 간에 네트워킹 서비스에 대한 부하를 분산합니다.

이 **Networker** 역할을 독립 실행형으로 만들려면 고유한 사용자 지정 컨트롤러 역할과 필요한 다른 역할을 만들 수 있습니다. 이를 통해 사용자 지정 역할에서 **roles_data** 파일을 쉽게 생성할 수 있습니다.

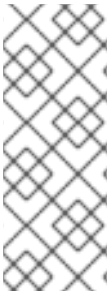
코어 **Heat** 템플릿 컬렉션의 디렉터리를 **stack** 사용자의 홈 디렉터리에 복사합니다.

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

이 디렉터리에서 사용자 지정 역할 파일을 추가하거나 수정합니다. 앞에서 언급한 역할 하위 명령과 함께 **--roles-path** 옵션을 사용하여 이 디렉터리를 사용자 지정 역할의 소스로 사용합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud roles generate -o my_roles_data.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

그러면 ~/ **roles** 디렉터리의 개별 역할에서 하나의 **my_roles _data.yaml** 파일이 생성됩니다.



참고

기본 역할 컬렉션에는 **ControllerOpenStack** 역할도 포함되어 있으며, 이 역할은 **Networker,Messaging** 및 **Database** 역할에 대한 서비스가 포함되지 않습니다. 독립 실행형 **Networker,Messaging** 및 **Database** 역할과 결합된 **ControllerOpenStack** 을 사용할 수 있습니다.

7.2.3. 지원되는 사용자 지정 역할

다음 표는 **/usr/share/openstack-tripleo-heat-templates/roles** 에서 사용할 수 있는 모든 지원되는 역할에 대해 설명합니다.

Role	설명	파일
BlockStorage	OpenStack Block Storage(cinder) 노드.	**BlockStorage.yaml**
CephAll	전체 독립 실행형 Ceph Storage 노드. OSD, MON, Object Gateway(RGW), Object Operations(MDS), Manager(MGR) 및 RBD Mirroring이 포함됩니다.	**CephAll.yaml**
CephFile	독립 실행형 스케일 아웃 Ceph Storage 파일 역할. OSD 및 MDS(Object Operations) 포함.	**CephFile.yaml**
CephObject	독립형 스케일 아웃 Ceph Storage 오브젝트 역할. OSD 및 개체 게이트웨이(RGW)를 포함합니다.	**CephObject.yaml**
CephStorage	Ceph Storage OSD 노드 역할.	**CephStorage.yaml**
ComputeAlt	대체 컴퓨팅 노드 역할.	**ComputeAlt.yaml**
ComputeDVR	DVR이 활성화된 컴퓨팅 노드 역할.	**ComputeDVR.yaml**
ComputeHCI	하이퍼컨버지드 인프라를 사용하는 계산 노드. 계산 및 Ceph OSD 서비스가 포함되어 있습니다.	**ComputeHCI.yaml**

Role	설명	파일
ComputeInstanceHA	Compute Instance HA 노드 역할. environment **/compute-instanceha.yaml 환경** 파일과 함께 를 사용합니다.	**ComputeInstanceHA .yaml**
ComputeLiquidio	Caviumio Smart NIC가 있는 계산 노드.	**ComputeLiquidio.ya ml**
ComputeOvsDpdkR T	Compute OVS DPDK RealTime 역할.	**ComputeOvsDpdkR T.yaml**
ComputeOvsDpdk	컴퓨팅 OVS DPDK 역할.	**ComputeOvsDpdk.y aml**
ComputePPC64LE	ppc64le 서버의 컴퓨팅 역할.	**ComputePPC64LE.y aml**
ComputeRealTime	실시간 동작을 위해 최적화된 컴퓨팅 역할. 이 역할을 사용하는 경우 **overcloud-realtime-compute** 이미지를 사용할 수 있어야 하며 역할별 매개 변수 **IsolCpusList** 및 **NovaVcpuPinSet** 을 실시간 계산 노드의 하드웨어에 따라 설정해야 합니다.	**ComputeRealTime.y aml**
ComputeSriovRT	컴퓨팅 SR-IOV RealTime 역할.	**ComputeSriovRT.ya ml**
ComputeSriov	컴퓨팅 SR-IOV 역할.	**ComputeSriov.yaml**
Compute	표준 컴퓨팅 노드 역할.	**compute.yaml**
ControllerAllNovaSta ndalone	데이터베이스, 메시징, 네트워킹 및 OpenStack 계산 (nova) 제어 구성 요소를 포함하지 않는 컨트롤러 역할. **Database**,**Messaging**,**Networker** 및 **Novacontrol** 역할과 함께 를 사용합니다.	**ControllerAllNovaSta ndalone.yaml**
ControllerNoCeph	핵심 컨트롤러 서비스가 로드되었지만 Ceph 스토리지 (MON) 구성 요소가 없는 컨트롤러 역할. 이 역할은 Ceph Storage 기능이 아닌 데이터베이스, 메시징 및 네트워크 기능을 처리합니다.	**ControllerNoCeph.ya ml**
ControllerNovaStand alone	OpenStack Compute(nova) 제어 구성 요소를 포함하지 않는 컨트롤러 역할. **Novacontrol** 역할과 함께 를 사용합니다.	**ControllerNovaStand alone.yaml**
ControllerOpenstack	데이터베이스, 메시징 및 네트워킹 구성 요소를 포함하지 않는 컨트롤러 역할. **Database**,**Messaging** 및 **Networker** 역할과 함께 를 사용합니다.	**ControllerOpenstack .yaml**

Role	설명	파일
ControllerStorageNfs	모든 핵심 서비스가 로드되고 Ceph NFS를 사용하는 컨트롤러 역할. 이 역할은 데이터베이스, 메시징 및 네트워크 기능을 처리합니다.	**ControllerStorageNfs.yaml**
컨트롤러	모든 핵심 서비스가 로드된 컨트롤러 역할. 이 역할은 데이터베이스, 메시징 및 네트워크 기능을 처리합니다.	**controller.yaml**
데이터베이스	독립 실행형 데이터베이스 역할. Pacemaker를 사용하여 Galera 클러스터로 관리되는 데이터베이스.	**Database.yaml**
HciCephAll	하이퍼컨버지드 인프라 및 모든 Ceph Storage 서비스가 포함된 계산 노드. OSD, MON, Object Gateway(RGW), Object Operations(MDS), Manager(MGR) 및 RBD Mirroring이 포함됩니다.	**HciCephAll.yaml**
HciCephFile	하이퍼컨버지드 인프라 및 Ceph Storage 파일 서비스가 포함된 계산 노드. OSD 및 MDS(Object Operations) 포함.	**HciCephFile.yaml**
HciCephMon	하이퍼컨버지드 인프라 및 Ceph Storage 블록 서비스가 포함된 계산 노드. OSD, MON, Manager가 포함됩니다.	**HciCephMon.yaml**
HciCephObject	하이퍼컨버지드 인프라 및 Ceph Storage 오브젝트 서비스가 포함된 계산 노드. OSD 및 개체 게이트웨이(RGW)를 포함합니다.	**HciCephObject.yaml**
IronicConductor	Ironic Conductor 노드 역할.	**IronicConductor.yaml**
메시징	독립 실행형 메시징 역할. Pacemaker로 관리되는 RabbitMQ.	**messaging.yaml**
Networker	독립 실행형 네트워킹 역할. OpenStack 네트워킹(neutron) 에이전트를 자체적으로 실행합니다. 배포에서 ML2/OVN 메커니즘 드라이버를 사용하는 경우 ML2/OVN을 사용하여 사용자 지정 역할 배포의 추가 단계를 참조하십시오.	**Networker.yaml**
Novacontrol	OpenStack Compute(nova) 제어 에이전트를 자체적으로 실행하는 독립 실행형 **nova-control** 역할.	**novacontrol.yaml**
ObjectStorage	Swift 오브젝트 스토리지 노드 역할.	**ObjectStorage.yaml**
Telemetry	모든 지표 및 알람 서비스가 포함된 원격 분석 역할.	**telemetry.yaml**

7.2.4. 역할 매개 변수 검사

각 역할은 다음 매개변수를 사용합니다.

name

(필수) 공백이나 특수 문자가 없는 일반 텍스트 이름인 역할의 이름입니다. 선택한 이름이 다른 리소스와 충돌하지 않는지 확인합니다. 예를 들어 **Network** 대신 **Network er** 를 사용합니다.

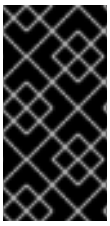
description

(선택 사항) 역할에 대한 일반 텍스트 설명입니다.

tags

(선택 사항) 역할 속성을 정의하는 **YAML** 태그 목록입니다. 이 매개변수를 사용하여 컨트롤러와 기본 태그 둘 다로 기본 역할을 정의합니다.

```
- name: Controller
  ...
  tags:
    - primary
    - controller
  ...
```



중요

기본 역할에 태그를 지정하지 않으면 정의된 첫 번째 역할이 기본 역할이 됩니다. 이 역할이 **Controller** 역할인지 확인합니다.

네트워크

역할에 구성할 네트워크의 **YAML** 목록입니다.

```
networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
```

기본 네트워크에는 **External,InternalApi,Storage, Storage Mgmt,Tenant** 및 **Management** 가 포함됩니다.

CountDefault

(선택 사항) 이 역할에 배포할 기본 노드 수를 정의합니다.

HostnameFormatDefault

(선택 사항) 역할에 대한 기본 호스트 이름 형식을 정의합니다. 기본 명명 규칙은 다음 형식을 사용합니다.

```
[STACK NAME]-[ROLE NAME]-[NODE ID]
```

예를 들어 기본 컨트롤러 노드의 이름은 다음과 같습니다.

```
overcloud-controller-0
overcloud-controller-1
overcloud-controller-2
...
```

disable_constraints

(선택 사항) **director**와 함께 배포할 때 **OpenStack Compute(nova)** 및 **OpenStack Image Storage(glance)** 제약 조건을 비활성화할지 여부를 정의합니다. 사전 프로비저닝된 노드가 있는 오버클라우드를 배포할 때 사용합니다. 자세한 내용은 ***Director*** *설치 및 사용 가이드* *의 "사전 프로비저닝된 노드를 사용하여 기본* ***Overcloud*** *구성"* *을 참조하십시오.*

disable_upgrade_deployment

(선택 사항) 특정 역할에 대한 업그레이드를 비활성화할지 여부를 정의합니다. 이렇게 하면 역할에서 개별 노드를 업그레이드하고 서비스 가용성을 보장할 수 있습니다. 예를 들어 **Compute** 및 **Swift Storage** 역할은 이 매개 변수를 사용합니다.

update_serial

(선택 사항) **OpenStack** 업데이트 옵션 중에 동시에 업데이트할 노드 수를 정의합니다. 기본 **roles_data.yaml** 파일에서 다음을 수행합니다.

- 기본값은 **Controller, Object Storage** 및 **Ceph Storage** 노드의 경우 **1** 입니다.
- **Compute** 및 **Block Storage** 노드의 기본값은 **25** 입니다.

사용자 지정 역할에서 이 매개변수를 생략하면 기본값은 **1** 입니다.

ServicesDefault

(선택 사항) 노드에 포함할 기본 서비스 목록을 정의합니다. 자세한 내용은 7.3.2절. “구성 가능한 서비스 아키텍처 검사”를 참조하십시오.

이러한 매개 변수는 새 역할을 생성하고 포함할 서비스를 정의하는 방법을 제공합니다.

openstack overcloud deploy 명령은 **roles_data** 파일의 매개 변수를 **Jinja2** 기반 템플릿에 통합합니다. 예를 들어 특정 지점에서 **overcloud.j2.yaml Heat** 템플릿은 **roles_data.yaml** 의 역할 목록을 반복하고 각 역할과 관련된 매개 변수와 리소스를 생성합니다.

overcloud.j2.yaml Heat 템플릿의 각 역할에 대한 리소스 정의가 다음 코드 조각으로 표시됩니다.

```
  {{role.name}}:
    type: OS::Heat::ResourceGroup
    depends_on: Networks
    properties:
      count: {get_param: {{role.name}}Count}
      removal_policies: {get_param: {{role.name}}RemovalPolicies}
      resource_def:
        type: OS::TripleO::{{role.name}}
        properties:
          CloudDomain: {get_param: CloudDomain}
          ServiceNetMap: {get_attr: [ServiceNetMap, service_net_map]}
          EndpointMap: {get_attr: [EndpointMap, endpoint_map]}
...
```

이 코드 조각은 **Jinja2** 기반 템플릿이 **{{role.name}}** 변수를 통합하여 각 역할의 이름을 **OS::Heat::ResourceGroup** 리소스로 정의하는 방법을 보여줍니다. 그런 다음 **roles_data** 파일의 각 **name** 매개변수를 사용하여 각각의 **OS::Heat::ResourceGroup** 리소스 이름을 지정합니다.

7.2.5. 새 역할 생성

이 예에서 목표는 **OpenStack** 대시보드(**Horizonizon**)만 호스팅하는 새 **Horizon** 역할을 생성하는 것입니다. 이 경우 새 역할 정보가 포함된 사용자 지정 역할 디렉터리를 생성합니다.

기본 역할 디렉터리의 사용자 지정 사본을 생성합니다.

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

~/roles/Horizon.yaml 이라는 새 파일을 생성하고 기본 및 핵심 **OpenStack** 대시보드 서비스를 포함하는 새 **Horizon** 역할을 만듭니다. 예를 들면 다음과 같습니다.

```
- name: Horizon
  CountDefault: 1
  HostnameFormatDefault: '%stackname%-horizon-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

또한 기본 **Overcloud**에 항상 **Horizon** 노드를 포함하도록 **CountDefault** 를 **1** 로 설정하는 것이 좋습니다.

기존 오버클라우드에서 서비스를 확장하는 경우 기존 서비스를 **Controller** 역할에 유지합니다. 새 오버클라우드를 생성하고 **OpenStack** 대시보드를 독립 실행형 역할에 유지하려면 컨트롤러 역할 정의에서 **OpenStack** 대시보드 구성 요소를 제거합니다.

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    ...
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    # - OS::TripleO::Services::Horizon              # Remove this service
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    ...
```

roles 디렉터리를 소스로 사용하여 새 **roles _data** 파일을 생성합니다.

```
$ openstack overcloud roles generate -o roles_data-horizon.yaml \
  --roles-path ~/roles \
  Controller Compute Horizon
```

특정 노드에 태그를 지정할 수 있도록 이 역할에 대한 새 플레이버를 정의해야 할 수 있습니다. 이 예에서는 다음 명령을 사용하여 **Horizon** 플레이버를 생성합니다.

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 horizon
$ openstack flavor set --property "cpu_arch"="x86_64" --property "capabilities:boot_option"="local" --
property "capabilities:profile"="horizon" horizon
$ openstack flavor set --property resources:VCPU=0 --property resources:MEMORY_MB=0 --
property resources:DISK_GB=0 --property resources:CUSTOM_BAREMETAL=1 horizon
```

다음 명령을 사용하여 노드를 새 플레이버에 태그합니다.

```
$ openstack baremetal node set --property capabilities='profile:horizon,boot_option:local' 58c3d07e-
24f2-48a7-bbb6-6843f0e8ee13
```

다음 환경 파일 스니펫을 사용하여 **Horizon** 노드 수 및 플레이버를 정의합니다.

```
parameter_defaults:
  OvercloudHorizonFlavor: horizon
  HorizonCount: 1
```

openstack overcloud deploy 명령을 실행할 때 새 **roles_data** 파일 및 환경 파일을 포함합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-horizon.yaml -e
~/templates/node-count-flavor.yaml
```

배포가 완료되면 컨트롤러 노드 **1**개, 컴퓨팅 노드 **1**개, **Networker** 노드 **1**개로 구성된 **3**노드 **Overcloud**가 생성됩니다. 오버클라우드의 노드 목록을 보려면 다음 명령을 실행합니다.

```
$ openstack server list
```

7.3. 구성 가능 서비스

7.3.1. 지침 및 제한 사항

구성 가능 노드 아키텍처에 대한 다음 지침 및 제한 사항을 확인합니다.

Pacemaker에서 관리하지 않는 서비스의 경우 다음을 수행합니다.

- 독립 실행형 사용자 정의 역할에 서비스를 할당할 수 있습니다.
- 초기 배포 후 추가 사용자 지정 역할을 생성하고 배포하여 기존 서비스를 확장할 수 있습니다.

Pacemaker에서 관리하는 서비스의 경우 다음을 수행합니다.

- 독립 실행형 사용자 정의 역할에 **Pacemaker** 관리 서비스를 할당할 수 있습니다.
- **Pacemaker**에는 노드 제한이 **16**개 있습니다. **Pacemaker** 서비스 **(OS::TripleO::Services::Pacemaker)**를 **16**개 노드에 할당하는 경우 후속 노드에서 **Pacemaker** 원격 서비스**(OS::TripleO::Services::PacemakerRemote)**를 대신 사용해야 합니다. 동일한 역할에 **Pacemaker** 서비스 및 **Pacemaker** 원격 서비스가 있을 수 없습니다.
- **Pacemaker** 관리 서비스를 포함하지 않는 역할에 **Pacemaker** 서비스 **(OS::TripleO::Services::Pacemaker)**를 포함하지 마십시오.
- **OS::TripleO::Services::Pacemaker** 또는 **OS::TripleO::Services: :Pacemaker Remote** 서비스가 포함된 사용자 지정 역할을 확장하거나 축소할 수 없습니다.

일반 제한 사항:

- 메이저 버전 업그레이드 중에 사용자 정의 역할 및 구성 가능 서비스를 변경할 수 없습니다.
- **Overcloud**를 배포한 후에는 역할에 대한 서비스 목록을 수정할 수 없습니다. **Overcloud** 배포 후 서비스 목록을 수정하면 배포 오류가 발생하고 노드에 서비스를 분리할 수 있습니다.

7.3.2. 구성 가능한 서비스 아키텍처 검사

코어 **heat** 템플릿 컬렉션에는 구성 가능 서비스 템플릿의 두 세트가 포함되어 있습니다.

- **Puppet/services** 에는 구성 가능 서비스를 구성하는 기본 템플릿이 포함되어 있습니다.

- **docker/services** 에는 주요 **OpenStack Platform** 서비스에 대한 컨테이너화된 템플릿이 포함되어 있습니다. 이러한 템플릿은 일부 기본 템플릿의 보강 기능을 하며 기본 템플릿으로 다시 참조합니다.

각 템플릿에는 목적을 식별하는 설명이 포함되어 있습니다. 예를 들어 **ntp.yaml** 서비스 템플릿에는 다음 설명이 포함되어 있습니다.

```
description: >
  NTP service deployment using puppet, this YAML file
  creates the interface between the HOT template
  and the puppet manifest that actually installs
  and configure NTP.
```

이러한 서비스 템플릿은 **RHOSP** 배포와 관련된 리소스로 등록됩니다. 즉, **overcloud-resource-registry-puppet.j2.yaml** 파일에 정의된 고유한 **heat** 리소스 네임스페이스를 사용하여 각 리소스를 호출할 수 있습니다. 모든 서비스는 리소스 유형에 **OS::TripleO::Services** 네임스페이스를 사용합니다.

일부 리소스는 기본 구성 가능 서비스 템플릿을 직접 사용합니다.

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: puppet/services/time/ntp.yaml
  ...
```

그러나 핵심 서비스에는 컨테이너가 필요하며 컨테이너화된 서비스 템플릿을 사용합니다. 예를 들어 **keystone** 컨테이너화된 서비스는 다음을 사용합니다.

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: docker/services/keystone.yaml
  ...
```

이러한 컨테이너화된 템플릿은 일반적으로 **Puppet** 구성을 포함하기 위해 기본 템플릿으로 다시 참조합니다. 예를 들어 **docker/services/keystone.yaml** 템플릿은 기본 템플릿의 출력을 **KeystoneBase** 매개변수에 저장합니다.

```
KeystoneBase:
  type: ../../puppet/services/keystone.yaml
```

컨테이너화된 템플릿은 기본 템플릿의 기능과 데이터를 통합할 수 있습니다.

overcloud.j2.yaml heat 템플릿에는 **roles_data.yaml** 파일에서 각 사용자 지정 역할에 대한 서비스 목록을 정의하는 **Jinja2** 기반 코드 섹션이 포함되어 있습니다.

```
{{role.name}}Services:
  description: A list of service resources (configured in the Heat
               resource_registry) which represent nested stacks
               for each service that should get installed on the {{role.name}} role.
  type: comma_delimited_list
  default: {{role.ServicesDefault|default([])}}
```

기본 역할의 경우 다음 서비스 목록 매개변수가 생성됩니다. **ControllerServices,ComputeServices,BlockStorageServices,ObjectStorageServices** 및 **CephStorageServices.**

roles_data.yaml 파일에서 각 사용자 지정 역할에 대한 기본 서비스를 정의합니다. 예를 들어 기본 **Controller** 역할에는 다음 내용이 포함됩니다.

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Core
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
...
```

그런 다음 이러한 서비스는 **ControllerServices** 매개변수의 기본 목록으로 정의됩니다.

환경 파일을 사용하여 서비스 매개변수의 기본 목록을 재정의할 수도 있습니다. 예를 들어 환경 파일에서 **ControllerServices** 를 **parameter_default** 로 정의하여 **roles_data.yaml** 파일의 서비스 목록을 재정의할 수 있습니다.

7.3.3. 역할에서 서비스 추가 및 제거

서비스를 추가하거나 제거하는 기본 방법은 노드 역할에 대한 기본 서비스 목록 복사본을 생성한 다음 서비스를 추가하거나 제거하는 것입니다. 예를 들어 컨트롤러 노드에서 **OpenStack Orchestration(heat)**을 제거하려고 할 수 있습니다. 이 경우 기본 역할 디렉터리의 사용자 지정 사본을 생성합니다.

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

~/roles/Controller.yaml 파일을 편집하고 **ServicesDefault** 매개변수의 서비스 목록을 수정합니다. **OpenStack Orchestration** 서비스로 스크롤하여 제거합니다.

```
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::HeatApi            # Remove this service
    - OS::TripleO::Services::HeatApiCfn         # Remove this service
    - OS::TripleO::Services::HeatApiCloudwatch  # Remove this service
    - OS::TripleO::Services::HeatEngine         # Remove this service
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::NeutronDhcpAgent
```

새 **roles_data** 파일을 생성합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud roles generate -o roles_data-no_heat.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

openstack overcloud deploy 명령을 실행할 때 이 새 **roles_data** 파일을 포함합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-no_heat.yaml
```

이렇게 하면 컨트롤러 노드에 **OpenStack Orchestration** 서비스가 설치되지 않고 **Overcloud**가 배포됩니다.

참고

또한 사용자 지정 환경 파일을 사용하여 **roles_data** 파일에서 서비스를 비활성화할 수 있습니다. 비활성화하도록 서비스를 리디렉션하여 **OS::Heat::None** 리소스로 전환합니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::Services::HeatApi: OS::Heat::None
  OS::TripleO::Services::HeatApiCfn: OS::Heat::None
  OS::TripleO::Services::HeatApiCloudwatch: OS::Heat::None
  OS::TripleO::Services::HeatEngine: OS::Heat::None
```

7.3.4. 비활성 서비스 활성화

일부 서비스는 기본적으로 비활성화되어 있습니다. 이러한 서비스는 **overcloud-resource-registry-puppet.j2.yaml** 파일에서 **null** 작업(**OS::Heat::None**)으로 등록됩니다. 예를 들어 블록 스토리지 백업 서비스(**cinder-backup**)가 비활성화됩니다.

```
  OS::TripleO::Services::CinderBackup: OS::Heat::None
```

이 서비스를 활성화하려면 **puppet/services** 디렉터리의 해당 **Heat** 템플릿에 리소스를 연결하는 환경 파일을 포함합니다. 일부 서비스에는 환경 디렉터리에 사전 정의된 환경 파일이 있습니다. 예를 들어 블록 스토리지 백업 서비스는 다음이 포함된 **environments/cinder-backup.yaml** 파일을 사용합니다.

```
resource_registry:
  OS::TripleO::Services::CinderBackup: ../docker/services/pacemaker/cinder-backup.yaml
...
```

이렇게 하면 기본 **null** 작업 리소스를 재정의하고 서비스를 활성화합니다. **openstack overcloud deploy** 명령을 실행할 때 이 환경 파일을 포함합니다.

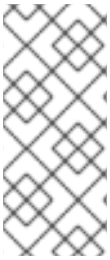
```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/cinder-backup.yaml
```

작은 정보

비활성화된 서비스를 활성화하는 방법에 대한 또 다른 예는 ***OpenStack Data Processing*** 가이드 의 설치를 참조하십시오. 이 섹션에서는 **Overcloud**에서 **OpenStack Data Processing** 서비스(**sahara**)를 활성화하는 방법에 대한 지침을 설명합니다.

7.3.5. 서비스가 없는 일반 노드 생성

Red Hat OpenStack Platform은 **OpenStack** 서비스를 구성하지 않고 일반 **Red Hat Enterprise Linux 7** 노드를 생성할 수 있는 기능을 제공합니다. 이는 핵심 **Red Hat OpenStack Platform** 환경 외부에서 소프트웨어를 호스팅해야 하는 경우에 유용합니다. 예를 들어 **OpenStack Platform**은 **Kibana** 및 **Sensu**와 같은 모니터링 툴과의 통합을 제공합니다(모니터링 툴 구성 가이드참조). **Red Hat**은 모니터링 툴 자체를 지원하지 않지만 **director**는 이러한 툴을 호스팅할 일반 **Red Hat Enterprise Linux 7** 노드를 생성할 수 있습니다.



참고

일반 노드에서는 기본 **Red Hat Enterprise Linux 7** 이미지가 아닌 기본 **overcloud-full** 이미지를 계속 사용합니다. 즉, 노드에 일부 **Red Hat OpenStack Platform** 소프트웨어가 설치되어 있지만 활성화 또는 구성되지 않습니다.

일반 노드를 생성하려면 **ServicesDefault** 목록 없이 새 역할이 필요합니다.

```
- name: Generic
```

사용자 지정 **roles_data** 파일(**roles_data_with_generic.yaml**)에 역할을 포함합니다. 기존 **Controller** 및 **Compute** 역할을 유지해야 합니다.

또한 환경 파일(**generic-node-params.yaml**)을 포함하여 필요한 일반 **Red Hat Enterprise Linux 7** 노드 수와 프로비저닝할 노드를 선택할 때 플레이버를 지정할 수도 있습니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  OvercloudGenericFlavor: baremetal
  GenericCount: 1
```

openstack overcloud deploy 명령을 실행할 때 역할 파일과 환경 파일을 모두 포함합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data_with_generic.yaml -e
~/templates/generic-node-params.yaml
```

이렇게 하면 컨트롤러 노드 **1**개, 컴퓨팅 노드 **1**개, 일반 **Red Hat Enterprise Linux 7** 노드 **1**개가 있는 **3-**노드 환경이 배포됩니다.

8장. 컨테이너화된 서비스

director는 핵심 OpenStack Platform 서비스를 오버클라우드에 컨테이너로 설치합니다. 이 섹션에서는 컨테이너화된 서비스의 작동 방식에 대한 몇 가지 배경 정보를 제공합니다.

8.1. 컨테이너화된 서비스 아키텍처

director는 핵심 OpenStack Platform 서비스를 오버클라우드에 컨테이너로 설치합니다. 컨테이너화된 서비스의 템플릿은 /usr/share/openstack-tripleo-heat-templates/docker/services/ 에 있습니다. 이러한 템플릿은 각 구성 가능 서비스 템플릿을 참조합니다. 예를 들어 OpenStack ID(keystone) 컨테이너화된 서비스 템플릿(docker/services/keystone.yaml)에는 다음 리소스가 포함되어 있습니다.

```
KeystoneBase:
  type: ../../puppet/services/keystone.yaml
  properties:
    EndpointMap: {get_param: EndpointMap}
    ServiceData: {get_param: ServiceData}
    ServiceNetMap: {get_param: ServiceNetMap}
    DefaultPasswords: {get_param: DefaultPasswords}
    RoleName: {get_param: RoleName}
    RoleParameters: {get_param: RoleParameters}
```

유형은 해당 OpenStack ID(keystone) 구성 가능 서비스를 참조하고 해당 템플릿에서 출력 데이터를 가져옵니다. 컨테이너화된 서비스는 이 데이터를 자체 컨테이너 특정 데이터와 병합합니다.

컨테이너화된 서비스를 사용하는 모든 노드는 OS::TripleO::Services::Docker 서비스를 활성화해야 합니다. 사용자 지정 역할 구성에 대한 roles_data.yaml 파일을 생성할 때 컨테이너화된 서비스로 기본 구성 가능 서비스를 사용하여 OS::TripleO::Services::Docker 서비스를 포함합니다. 예를 들어 Keystone 역할은 다음 역할 정의를 사용합니다.

```
- name: Keystone
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
```

```
- OS::TripleO::Services::MySQLClient
- OS::TripleO::Services::Docker
- OS::TripleO::Services::Keystone
```

8.2. 컨테이너화된 서비스 매개변수

컨테이너화된 각 서비스 템플릿에는 **director**의 **OpenStack Orchestration(heat)** 서비스에 전달되는 데이터 집합을 정의하는 **outputs** 섹션이 포함되어 있습니다. 표준 구성 가능 서비스 매개변수(7.2.4절. “역할 매개 변수 검사”참조) 외에도 템플릿에는 컨테이너 구성과 관련된 매개변수 세트가 포함되어 있습니다.

puppet_config

서비스를 구성할 때 **Puppet**에 전달할 데이터입니다. 초기 오버클라우드 배포 단계에서 **director**는 실제 컨테이너화된 서비스가 실행되기 전에 서비스를 구성하는 데 사용되는 컨테이너 세트를 생성합니다. 이 매개 변수에는 다음 하위 매개 변수가 포함됩니다. +

- **config_volume -** 구성을 저장하는 마운트된 **Docker** 볼륨입니다.
- **puppet_tags -** 구성 중에 **Puppet**에 전달할 태그입니다. 이러한 태그는 **OpenStack Platform**에서 **Puppet** 실행을 특정 서비스의 구성 리소스로 제한하는 데 사용됩니다. 예를 들어 **OpenStack ID(keystone)** 컨테이너화된 서비스는 **keystone_config** 태그를 사용하여 구성 컨테이너에서 필요한 모든 **keystone_config Puppet** 리소스만 실행되도록 합니다.
- **step_config - Puppet**에 전달되는 구성 데이터입니다. 일반적으로 이는 참조된 구성 가능 서비스에서 상속됩니다.
- **config_image -** 서비스를 구성하는 데 사용되는 컨테이너 이미지입니다.

kolla_config

구성 파일 위치, 디렉터리 권한 및 컨테이너에서 실행할 명령을 정의하여 서비스를 시작하는 컨테이너별 데이터 집합입니다.

docker_config

서비스의 구성 컨테이너에서 실행할 작업. 모든 작업은 **director**에서 스테이징된 배포를 수행하는 데 도움이 되는 단계로 그룹화됩니다. 단계는 다음과 같습니다. +

- **1단계 -** 로드 밸런서 구성

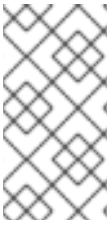
- 2단계 - 핵심 서비스(데이터베이스, Redis)
- 3 단계 - OpenStack Platform 서비스의 초기 구성
- 4 단계 - 일반 OpenStack Platform 서비스 구성
- 5단계 - 서비스 활성화

host_prep_tasks

컨테이너화된 서비스를 수용하도록 베어 메탈 노드에 대한 작업 준비.

8.3. OPENSTACK PLATFORM 컨테이너 수정

Red Hat은 Red Hat Container Catalog(registry.redhat.io)를 통해 사전 빌드된 컨테이너 이미지 세트를 제공합니다. 이러한 이미지를 수정하고 계층을 추가할 수 있습니다. 이는 인증된 타사 드라이버의 RPM을 컨테이너에 추가하는 데 유용합니다.



참고

수정된 OpenStack Platform 컨테이너 이미지에 대한 지속적인 지원을 보장하려면 결과 이미지가 "Red Hat Container Support Policy" 를 준수하는지 확인하십시오.

이 예에서는 최신 openstack-keystone 이미지를 사용자 지정하는 방법을 보여줍니다. 그러나 이러한 지침은 다른 이미지에도 적용할 수 있습니다.

1. 수정하려는 이미지를 가져옵니다. 예를 들어 openstack-keystone 이미지의 경우 다음을 수행합니다.

```
$ sudo docker pull registry.redhat.io/rhosp13/openstack-keystone:latest
```

2. 원본 이미지의 기본 사용자를 확인합니다. 예를 들어 openstack-keystone 이미지의 경우 다음을 수행합니다.

```
$ sudo docker run -it registry.redhat.io/rhosp13/openstack-keystone:latest whoami
root
```

참고

openstack-keystone 이미지는 **root** 를 기본 사용자로 사용합니다. 다른 이미지에서는 특정 사용자를 사용합니다. 예를 들어 **openstack-glance-api** 는 기본 사용자에게 **glance** 를 사용합니다.

3. **Dockerfile** 을 생성하여 기존 컨테이너 이미지에 추가 계층을 빌드합니다. 다음은 **Container Catalog**에서 최신 **OpenStack Identity(keystone)** 이미지를 가져와서 사용자 지정 **RPM** 파일을 이미지에 설치하는 예입니다.

```
FROM registry.redhat.io/rhosp13/openstack-keystone
MAINTAINER Acme
LABEL name="rhosp13/openstack-keystone-acme" vendor="Acme" version="2.1"
release="1"

# switch to root and install a custom RPM, etc.
USER root
COPY custom.rpm /tmp
RUN rpm -ivh /tmp/custom.rpm

# switch the container back to the default user
USER root
```

4. 새 이미지를 빌드하고 태그를 지정합니다. 예를 들어 **/home/stack/keystone** 디렉터리에 로컬 **Dockerfile** 을 저장하고 언더클라우드의 로컬 레지스트리에 태그를 지정하려면 다음을 수행합니다.

```
$ docker build /home/stack/keystone -t "192.168.24.1:8787/rhosp13/openstack-keystone-
acme:rev1"
```

5. 결과 이미지를 언더클라우드의 로컬 레지스트리로 내보냅니다.

```
$ docker push 192.168.24.1:8787/rhosp13/openstack-keystone-acme:rev1
```

6. 오버클라우드 컨테이너 이미지 환경 파일(일반적으로 **overcloud_images.yaml**)을 편집하고 사용자 지정 컨테이너 이미지를 사용하도록 적절한 매개변수를 변경합니다.

주의

Container Catalog는 전체 소프트웨어 스택이 빌드된 컨테이너 이미지를 게시합니다. **Container Catalog**에서 업데이트 및 보안 수정 사항으로 컨테이너 이미지를 릴리스하면 기존 사용자 정의 컨테이너에 이러한 업데이트가 포함되지 않으므로 카탈로그에서 새 이미지 버전을 사용하여 다시 빌드해야 합니다.

8.4. 벤더 플러그인 배포

타사 하드웨어를 블록 스토리지 백엔드로 사용하려면 벤더 플러그인을 배포해야 합니다. 다음 예제에서는 **Dell EMC** 하드웨어를 블록 스토리지 백엔드로 사용하기 위해 벤더 플러그인을 배포하는 방법을 보여줍니다.

1. **registry.connect.redhat.com** 카탈로그에 로그인합니다.

```
$ docker login registry.connect.redhat.com
```

2. 플러그인을 다운로드합니다.

```
$ docker pull registry.connect.redhat.com/dellemc/openstack-cinder-volume-dellemc-rhosp13
```

3. **OpenStack** 배포와 관련된 언더클라우드 **IP** 주소를 사용하여 이미지를 로컬 언더클라우드 레지스트리로 태그하고 푸시합니다.

```
$ docker tag registry.connect.redhat.com/dellemc/openstack-cinder-volume-dellemc-rhosp13
192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-rhosp13

$ docker push 192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-rhosp13
```

4. 다음 매개변수가 포함된 추가 환경 파일을 사용하여 오버클라우드를 배포합니다.

```
parameter_defaults:
  DockerCinderVolumeImage: 192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-
rhosp13
```

9장. 기본 네트워크 격리

이 장에서는 표준 네트워크 격리 구성을 사용하여 오버클라우드를 구성하는 방법을 보여줍니다. 여기에는 다음이 포함됩니다.

- 네트워크 격리(**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**)를 활성화하기 위해 렌더링된 환경 파일입니다.
- 네트워크 기본값을 구성하는 복사된 환경 파일(**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**).
- **IP** 범위, 서브넷, 가상 **IP** 등의 네트워크 설정을 정의하는 **network _data** 파일. 이 예에서는 기본값의 복사본을 생성하고 자체 네트워크에 맞게 편집하는 방법을 보여줍니다.
- 각 노드의 **NIC** 레이아웃을 정의하는 템플릿입니다. 오버클라우드 코어 템플릿 컬렉션에는 다양한 사용 사례에 대한 기본값 세트가 포함되어 있습니다.
- **NIC**를 활성화하는 환경 파일입니다. 이 예에서는 **environment** 디렉터리에 있는 기본 파일을 사용합니다.
- 네트워킹 매개 변수를 사용자 지정할 추가 환경 파일입니다.

참고

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

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

이 장의 다음 내용은 이러한 각 측면을 정의하는 방법을 보여줍니다.

9.1. 네트워크 격리

Overcloud는 기본적으로 **provisioning** 네트워크에 서비스를 할당합니다. 그러나 **director**는 오버클라우드 네트워크 트래픽을 격리된 네트워크로 나눌 수 있습니다. 분리된 네트워크를 사용하기 위해 오버클라우드에는 이 기능을 활성화하는 환경 파일이 포함되어 있습니다. 코어 **heat** 템플릿의 **environments/network-isolation.j2.yaml** 파일은 구성 가능한 네트워크 파일에 있는 각 네트워크의 모든 포트 및 **VIP**를 정의하는 **Jinja2** 파일입니다. 렌더링되면 **network-isolation.yaml** 파일이 전체 리소스 레지스트리와 동일한 위치에 생성됩니다.

```
resource_registry:
  # networks as defined in network_data.yaml
  OS::TripleO::Network::Storage: ../network/storage.yaml
  OS::TripleO::Network::StorageMgmt: ../network/storage_mgmt.yaml
  OS::TripleO::Network::InternalApi: ../network/internal_api.yaml
  OS::TripleO::Network::Tenant: ../network/tenant.yaml
  OS::TripleO::Network::External: ../network/external.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::StorageVipPort: ../network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: ../network/ports/storage_mgmt.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: ../network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::ExternalVipPort: ../network/ports/external.yaml
  OS::TripleO::Network::Ports::RedisVipPort: ../network/ports/vip.yaml

  # Port assignments by role, edit role definition to assign networks to roles.
  # Port assignments for the Controller
  OS::TripleO::Controller::Ports::StoragePort: ../network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: ../network/ports/storage_mgmt.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: ../network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::TenantPort: ../network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ExternalPort: ../network/ports/external.yaml

  # Port assignments for the Compute
  OS::TripleO::Compute::Ports::StoragePort: ../network/ports/storage.yaml
  OS::TripleO::Compute::Ports::InternalApiPort: ../network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::TenantPort: ../network/ports/tenant.yaml

  # Port assignments for the CephStorage
  OS::TripleO::CephStorage::Ports::StoragePort: ../network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: ../network/ports/storage_mgmt.yaml
```

이 파일의 첫 번째 섹션에는 **OS::TripleO::Network::*** 리소스에 대한 리소스 레지스트리 선언이 있습니다. 기본적으로 이러한 리소스는 네트워크를 생성하지 않는 **OS::Heat::None** 리소스 유형을 사용합니다. 이러한 리소스를 각 네트워크의 **YAML** 파일로 리디렉션하면 이러한 네트워크를 생성할 수 있습니다.

다음 여러 섹션에서는 각 역할의 노드의 **IP** 주소를 생성합니다. 컨트롤러 노드에는 각 네트워크에 **IP**가 있습니다. 계산 및 스토리지 노드에는 각각 네트워크의 하위 집합에 **IP**가 있습니다.

10장. *사용자 지정 구성 가능 네트워크* 및 **11**장. *사용자 정의 네트워크 인터페이스 템플릿* 과 같은 오버클라우드 네트워킹의 기타 기능은 이 네트워크 격리 환경 파일을 사용합니다. 따라서 배포 명령을 사용하여 렌더링된 파일의 이름을 포함해야 합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  ...
```

9.2. 격리된 네트워크 구성 수정

기본 **network_data.yaml** 파일을 복사하고 복사본을 수정하여 기본 격리된 네트워크를 구성합니다.

절차

1. 기본 **network_data** 파일을 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/network_data.yaml /home/stack/.
```

2. **network_data.yaml** 파일의 로컬 사본을 편집하고 네트워킹 요구 사항에 맞게 매개변수를 수정합니다. 예를 들어 내부 **API** 네트워크에는 다음과 같은 기본 네트워크 세부 정보가 포함되어 있습니다.

```
- name: InternalApi
  name_lower: internal_api
  vip: true
  vlan: 201
  ip_subnet: '172.16.2.0/24'
  allocation_pools: [{'start': '172.16.2.4', 'end': '172.16.2.250'}]
```

각 네트워크에 대해 다음을 편집합니다.

- **VLAN** 은 이 네트워크에 사용할 **VLAN ID**를 정의합니다.
- **ip_subnet** 및 **ip_allocation_pools** 는 네트워크의 기본 서브넷 및 **IP** 범위를 설정합니다.
- 게이트웨이는 네트워크의 게이트웨이를 설정합니다. 주로 외부 네트워크의 기본 경로를 정의하는 데 사용되지만 필요한 경우 다른 네트워크에 사용할 수 있습니다.

n 옵션을 사용하여 배포와 함께 사용자 지정 **network_data** 파일을 포함합니다. **n** 옵션을 사용하지 않으면 배포 명령에서 기본 네트워크 세부 정보를 사용합니다.

9.3. 네트워크 인터페이스 템플릿

Overcloud 네트워크 설정에는 네트워크 인터페이스 템플릿 집합이 필요합니다. 이러한 템플릿은 **YAML** 형식의 표준 **heat** 템플릿입니다. 각 역할에는 **director**가 해당 역할 내의 각 노드를 올바르게 구성할 수 있도록 **NIC** 템플릿이 필요합니다.

모든 **NIC** 템플릿에는 표준 **Heat** 템플릿과 동일한 섹션이 포함됩니다.

heat_template_version

사용할 구문 버전입니다.

description

템플릿의 문자열 설명입니다.

parameters

템플릿에 포함할 네트워크 매개 변수입니다.

resources

매개 변수에 정의된 매개 변수를 사용하여 네트워크 구성 스크립트에 적용합니다.

출력

구성에 사용되는 최종 스크립트를 렌더링합니다.

/usr/share/openstack-tripleo-heat-templates/network/config 의 기본 **NIC** 템플릿은 **Jinja2** 구문을 활용하여 템플릿을 렌더링하는 데 도움이 됩니다. 예를 들어 **single-nic-vlans** 구성의 다음 코드 조각은 각 네트워크에 대해 일련의 **VLAN**을 렌더링합니다.

```
{%- for network in networks if network.enabled|default(true) and network.name in role.networks %}
- type: vlan
  vlan_id:
    get_param: {{network.name}}NetworkVlanID
  addresses:
  - ip_netmask:
      get_param: {{network.name}}IpSubnet
{%- if network.name in role.default_route_networks %}
```

기본 컴퓨팅 노드의 경우 스토리지, 내부 **API** 및 테넌트 네트워크에 대한 네트워크 정보만 렌더링합니다.

```
- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
- type: vlan
  vlan_id:
    get_param: TenantNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: TenantIpSubnet
```

11장. *사용자 정의 네트워크 인터페이스 템플릿* 기본 Jinja2 기반 템플릿을 사용자 지정의 기준으로 사용할 수 있는 표준 YAML 버전으로 렌더링하는 방법을 살펴봅니다.

9.4. 기본 네트워크 인터페이스 템플릿

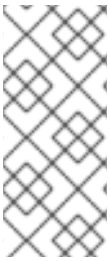
director에는 대부분의 일반적인 네트워크 시나리오에 맞게 /usr/share/openstack-tripleo-heat-templates/network/config/ 에 템플릿이 포함되어 있습니다. 다음 표에서는 템플릿을 활성화하는 데 사용해야 하는 각 NIC 템플릿 세트와 해당 환경 파일을 간략하게 설명합니다.

참고

NIC 템플릿을 활성화하는 각 환경 파일은 접미사 .j2.yaml 을 사용합니다. 렌더링되지 않은 Jinja2 버전입니다. 배포에. yaml 접미사만 사용하는 렌더링된 파일 이름을 포함해야 합니다.

NIC 디렉토리	설명	환경 파일
single-nic-vlans	**컨트롤**플레인 및 VLAN이 기본 Open vSwitch 브리지에 연결된 단일 NIC(nic1).	**environments/net-single-nic-with-vlans.j2.yaml**
single-nic-linux-bridge-vlans	기본 Linux 브리지에**컨트롤**플레인 및 VLAN이 연결된 단일 NIC(nic1)입니다.	**environments/net-single-nic-linux-bridge-with-vlans**

NIC 디렉토리	설명	환경 파일
bond-with-vlans	**nic1** 에 연결된 컨트롤 플레인. 본딩된 NIC 구성(nic**2 및 nic 3) 및 연결된 VLAN이 있는**기본 Open vSwitch 브리지.	**environments/net-bond-with-vlans.yaml**
multiple-nics	**nic1** 에 연결된 컨트롤 플레인. 순차적으로 각 NIC를 network **_data 파일에 정의된 각 네트워크에** 할당합니다. 기본적으로 Storage to **nic2**, Storage Management to **nic3**, Internal API to **nic4**, **br-tenant** 브리지에서 **nic5** 에 테넌트, 기본 Open vSwitch 브리지에서 **nic6** 으로 외부입니다.	**environments/net-multiple-nics.yaml**



참고

외부 네트워크(예: **net-bond-with-vlans-no-external.yaml)** 및 **IPv6(**예: **net-bond-with-vlans-v6.yaml)**를 사용하기 위한 환경 파일이 있습니다. 이러한 인터페이스는 이전 버전과의 호환성을 위해 제공되며 구성 가능한 네트워크에서 작동하지 않습니다.

각 기본 **NIC** 템플릿 세트에는 **role.role.j2.yaml** 템플릿이 포함되어 있습니다. 이 파일은 **Jinja2**를 사용하여 구성 가능 역할에 대한 추가 파일을 렌더링합니다. 예를 들어 오버클라우드에서 **Compute, Controller, Ceph Storage** 역할을 사용하는 경우 배포에서 **role.role.j2.yaml** 을 기반으로 새 템플릿을 렌더링합니다(예:).

- **compute.yaml**
- **controller.yaml**
- **ceph-storage.yaml.**

9.5. 기본 네트워크 격리 활성화

다음 절차에서는 기본 **NIC** 템플릿 중 하나를 사용하여 기본 네트워크 격리를 활성화하는 방법을 보여줍니다. 이 경우 **VLANs** 템플릿**(Single-nic-vlans)**이 있는 단일**NIC**입니다.

절차

1. **openstack overcloud deploy** 명령을 실행할 때 에 대해 렌더링된 환경 파일 이름을 포함해야 합니다.

   - 사용자 지정 **network_data** 파일.
   - 기본 네트워크 격리의 렌더링된 파일 이름입니다.
   - 렌더링된 기본 네트워크 환경 파일의 파일 이름입니다.
   - 렌더링된 기본 네트워크 인터페이스 구성의 파일 이름
   - 구성과 관련된 추가 환경 파일입니다.

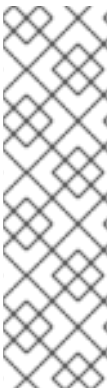
예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  ...
  -n /home/stack/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
  ...
```

10장. 사용자 지정 구성 가능 네트워크

이 장에서는 9장. *기본 네트워크 격리* 에 설명된 개념 및 절차에서 설명하고, 추가 구성 가능 네트워크를 사용하여 오버클라우드를 구성하는 방법을 보여줍니다. 여기에는 다음이 포함됩니다.

- 네트워크 격리를 활성화하는 환경 파일(/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml).
- 네트워크 기본값(/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml)을 구성하는 환경 파일입니다.
- 기본값 외부에 추가 네트워크를 생성하는 사용자 지정 **network_data** 파일.
- 역할에 사용자 지정 네트워크를 할당할 사용자 지정 **roles_data** 파일입니다.
- 각 노드의 **NIC** 레이아웃을 정의하는 템플릿입니다. 오버클라우드 코어 템플릿 컬렉션에는 다양한 사용 사례에 대한 기본값 세트가 포함되어 있습니다.
- **NIC**를 활성화하는 환경 파일입니다. 이 예에서는 환경 디렉터리에 있는 기본 파일을 사용합니다.
- 네트워킹 매개 변수를 사용자 지정할 추가 환경 파일입니다. 이 예에서는 환경 파일을 사용하여 구성 가능한 네트워크에 대한 **OpenStack** 서비스 매핑을 사용자 지정합니다.



참고

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

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

이 장의 다음 내용은 이러한 각 측면을 정의하는 방법을 보여줍니다.

10.1. 구성 가능 네트워크

오버클라우드는 기본적으로 다음 사전 정의된 네트워크 세그먼트 세트를 사용합니다.

- 컨트롤 플레인
- 내부 API
- 스토리지
- 스토리지 관리
- 테넌트
- 외부
- 관리 (선택 사항)

구성 가능 네트워크를 사용하면 다양한 서비스에 대한 네트워크를 추가할 수 있습니다. 예를 들어 NFS 트래픽 전용 네트워크가 있는 경우 여러 역할에 제공할 수 있습니다.

director는 배포 및 업데이트 단계에서 사용자 지정 네트워크 생성을 지원합니다. 이러한 추가 네트워크는 Ironic 베어 메탈 노드, 시스템 관리 또는 다양한 역할에 맞는 별도의 네트워크를 생성하는 데 사용할 수 있습니다. 또한 트래픽이 네트워크 간에 라우팅되는 분할 배포를 위해 여러 네트워크 세트를 생성할 수도 있습니다.

단일 데이터 파일(network_data.yaml)은 배포할 네트워크 목록을 관리합니다. 이 파일을 -n 옵션을 사용하여 배포 명령에 포함합니다. 이 옵션을 사용하지 않으면 배포 시 기본 파일(/usr/share/openstack-tripleo-heat-templates/network_data.yaml)을 사용합니다.

10.2. 구성 가능 네트워크 추가

구성 가능 네트워크를 사용하여 다양한 서비스의 네트워크를 추가합니다. 예를 들어 스토리지 백업 트

래픽 전용 네트워크가 있는 경우 네트워크를 여러 역할에 제공할 수 있습니다.

절차

1. 기본 **network_data** 파일을 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/network_data.yaml /home/stack/.
```

2. **network_data.yaml** 파일의 로컬 사본을 편집하고 새 네트워크의 섹션을 추가합니다. 예를 들면 다음과 같습니다.

```
- name: StorageBackup
  vip: true
  name_lower: storage_backup
  ip_subnet: '172.21.1.0/24'
  allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
  gateway_ip: '172.21.1.1'
```

- 사람이 읽을 수 있는 네트워크의 이름을 설정합니다. 이 매개변수는 유일한 필수 매개변수입니다. 가독성을 위해 이름을 정규화하려면 **name_lower** 매개변수를 사용합니다(예: **InternalApi** 를 **internal_api** 로 변경하려는 경우). **name** 매개 변수를 수정하지 마십시오.

- **VIP: true** 는 새 네트워크에 가상 **IP** 주소(VIP)를 만듭니다. 이 **IP**는**Service-to-network** 매핑 매개 변수(**Service-to-network mapping** 매개변수)에 나열된 서비스의 대상 **IP**로 사용됩니다. **VIP**는 **Pacemaker**를 사용하는 역할에서만 사용됩니다. **Overcloud**의 부하 분산 서비스는 이러한 **IP**의 트래픽을 해당 서비스 엔드포인트로 리디렉션합니다.

- **ip_subnet,allocation_pools** 및 **gateway_ip** 는 네트워크의 기본 **IPv4** 서브넷, **IP** 범위 및 게이트웨이를 설정합니다.

n 옵션을 사용하여 배포와 함께 사용자 지정 **network_data** 파일을 포함합니다. **n** 옵션을 사용하지 않으면 배포 명령에서 기본 네트워크 집합을 사용합니다.

10.3. 역할에 구성 가능한 네트워크 포함

구성 가능 네트워크를 사용자 환경에 정의된 오버클라우드 역할에 할당할 수 있습니다. 예를 들어 **Ceph Storage** 노드에 사용자 지정 **StorageBackup** 네트워크를 포함할 수 있습니다.

절차

1. **custon roles_data** 파일이 아직 없는 경우 기본값을 홈 디렉터리에 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml /home/stack/.
```

2. 사용자 지정 **roles_data** 파일을 편집합니다.

3. 구성 가능 네트워크를 추가하려는 역할로 스크롤하고 네트워크 이름을 네트워크 목록에 추가합니다 **.** 예를 들어 네트워크를 **Ceph Storage** 역할에 추가하려면 다음 스니펫을 가이드로 사용합니다.

```
- name: CephStorage
  description: |
    Ceph OSD Storage node role
  networks:
    - Storage
    - StorageMgmt
    - StorageBackup
```

4. 사용자 지정 네트워크를 해당 역할에 추가한 후 파일을 저장합니다.

openstack overcloud deploy 명령을 실행할 때 **-r** 옵션을 사용하여 **roles_data** 파일을 포함합니다**. r** 옵션을 사용하지 않으면 배포 명령은 해당 할당된 네트워크에 기본 역할 세트를 사용합니다.

10.4. 구성 가능한 네트워크에 OPENSTACK 서비스 할당

각 **OpenStack** 서비스는 리소스 레지스트리의 기본 네트워크 유형에 할당됩니다**.** 이러한 서비스는 네트워크 유형의 할당된 네트워크 내의 **IP** 주소에 바인딩됩니다**. OpenStack** 서비스는 이러한 네트워크 간에 나뉩니다**.** 실제 실제 네트워크는 네트워크 환경 파일에 정의된 대로 다를 수 있습니다**.** 환경 파일에서 새 네트워크 맵**(**예**: /home/stack/templates/service-reassignments.yaml)**을 정의하여 **OpenStack** 서비스를 다른 네트워크 유형에 다시 할당할 수 있습니다**. ServiceNetMap** 매개변수는 각 서비스에 사용할 네트워크 유형을 결정합니다.

예를 들어 강조 표시된 섹션을 수정하여 스토리지 관리 네트워크 서비스를 스토리지 백업 네트워크에 다시 할당할 수 있습니다.

```
parameter_defaults:
  ServiceNetMap:
    SwiftMgmtNetwork: storage_backup
    CephClusterNetwork: storage_backup
```

이러한 매개 변수를 **storage_backup** 으로 변경하면 스토리지 관리 네트워크가 아닌 스토리지 백업 네트워크에 이러한 서비스가 수행됩니다. 즉, 스토리지 관리 네트워크가 아니라 스토리지 백업 네트워크에 대해 **parameter_defaults** 세트를 정의하면 됩니다.

director는 사용자 정의 **ServiceNetMap** 매개변수 정의를 **ServiceNetMapDefaults** 에서 가져온 기본값을 미리 정의된 목록으로 병합하고 기본값을 재정의합니다. 그런 다음 **director**는 다양한 서비스에 대한 네트워크 할당을 구성하는 데 사용되는 사용자 지정을 포함하는 전체 목록을 반환합니다.

서비스 매핑은 **Pacemaker**를 사용하는 노드의 경우 **network_data** 파일에서 **vip: true** 를 사용하는 네트워크에만 적용됩니다. 오버클라우드의 로드 밸런서는 **VIP**의 트래픽을 특정 서비스 엔드포인트로 리디렉션합니다.

참고

전체 기본 서비스 목록은 **/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** 내의 **ServiceNetMapDefaults** 매개변수에서 확인할 수 있습니다.

10.5. 사용자 지정 구성 가능 네트워크 활성화

기본 **NIC** 템플릿 중 하나를 사용하여 사용자 지정 구성 가능한 네트워크를 활성화합니다. 이 예제에서는 **VLANs** 템플릿**(net-single-nic-with-vlans)**과 함께 **Single NIC**를 사용합니다.

절차

1. **openstack overcloud deploy** 명령을 실행할 때 다음을 포함해야 합니다.

   - 사용자 지정 **network_data** 파일.
   - 네트워크**-**역할 할당이 있는 사용자 지정 **roles_data** 파일.
   - 렌더링된 기본 네트워크 격리 구성의 파일 이름입니다.
   - 렌더링된 기본 네트워크 환경 파일의 파일 이름입니다.

- 렌더링된 기본 네트워크 인터페이스 구성의 파일 이름입니다.
- 서비스 재할당과 같은 네트워크 관련 추가 환경 파일.

예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  ...
  -n /home/stack/network_data.yaml \
  -r /home/stack/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
  -e /home/stack/templates/service-reassignments.yaml \
  ...
```

이렇게 하면 추가 사용자 지정 네트워크를 포함하여 오버클라우드의 노드에 구성 가능한 네트워크가 배포됩니다.

중요

관리 네트워크와 같은 새 사용자 지정 네트워크를 도입하는 경우 템플릿을 다시 렌더링해야 합니다. **roles_data.yaml** 파일에 네트워크 이름을 추가하는 것만으로는 충분하지 않습니다.

10.6. 기본 네트워크 이름 변경

network_data.yaml 파일을 사용하여 기본 네트워크의 사용자가 볼 수 있는 이름을 수정할 수 있습니다.

- **InternalApi**
- 외부
- 스토리지

- **StorageMgmt**

- **테넌트**

이러한 이름을 변경하려면 name 필드를 수정하지 마십시오. 대신 name_lower 필드를 네트워크의 새 이름으로 변경하고 새 이름으로 ServiceNetMap을 업데이트합니다.

절차

1. **network_data.yaml 파일에서 이름을 변경할 각 네트워크의 name_lower 매개변수에 새 이름을 입력합니다.**

```
- name: InternalApi
  name_lower: MyCustomInternalApi
```

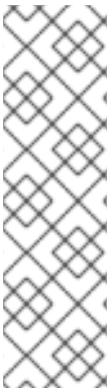
2. **service_net_map _replace 매개변수에 name_ lower 매개변수의 기본값을 포함합니다.**

```
- name: InternalApi
  name_lower: MyCustomInternalApi
  service_net_map_replace: internal_api
```

11장. 사용자 정의 네트워크 인터페이스 템플릿

이 장에서는 9장. *기본 네트워크 격리* 에 설명된 개념과 절차에 대해 설명합니다. 이 장의 목적은 사용자 환경의 노드에 맞게 사용자 지정 네트워크 인터페이스 템플릿 세트를 생성하는 방법을 보여줍니다. 여기에는 다음이 포함됩니다.

- 네트워크 격리를 활성화하는 환경 파일(**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**).
- 네트워크 기본값(**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**)을 구성하는 환경 파일입니다.
- 각 노드의 **NIC** 레이아웃을 정의하는 템플릿입니다. 오버클라우드 코어 템플릿 컬렉션에는 다양한 사용 사례에 대한 기본값 세트가 포함되어 있습니다. 이 경우 사용자 지정 템플릿의 기본 사항을 렌더링합니다.
- **NIC**를 활성화하는 사용자 지정 환경 파일입니다. 이 예에서는 사용자 지정 인터페이스 템플릿을 참조하는 사용자 지정 환경 파일(**/home/stack/templates/custom-network-configuration.yaml**)을 사용합니다.
- 네트워킹 매개 변수를 사용자 지정할 추가 환경 파일입니다.
- 네트워크를 사용자 지정하는 경우 사용자 지정 **network_data** 파일.
- 추가 또는 사용자 지정 구성 가능한 네트워크를 생성하는 경우 사용자 지정 **network_data** 파일 및 사용자 지정 **roles_data** 파일.



참고

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

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

11.1. 사용자 지정 네트워크 아키텍처

기본 NIC 템플릿이 특정 네트워크 구성에 적합하지 않을 수 있습니다. 예를 들어 특정 네트워크 레이아웃에 맞는 자체 사용자 지정 NIC 템플릿을 생성할 수 있습니다. 에서 제어 서비스와 데이터 서비스를 분리하여 NIC를 분리할 수 있습니다. 이 경우 다음과 같은 방식으로 서비스를 NIC 할당에 매핑할 수 있습니다.

- NIC1 (프로비저닝):
  - 프로비저닝/컨트롤 플레인
- NIC2 (Control Group)
  - 내부 API
  - 스토리지 관리
  - 외부(공용 API)
- NIC3 (데이터 그룹)
  - 테넌트 네트워크(VXLAN 터널링)
  - 테넌트 VLAN/프로바이더 VLAN
  - 스토리지
  - 외부 VLAN (유동 IP/SNAT)
- NIC4 (관리)

- 관리

11.2. 사용자 정의를 위한 기본 네트워크 인터페이스 템플릿 렌더링

사용자 지정 인터페이스 템플릿 구성을 간소화하려면 기본 **NIC** 템플릿의 **Jinja2** 구문을 렌더링하고 사용자 지정 구성의 기반으로 렌더링된 템플릿을 사용합니다.

절차

1. **process-templates.py** 스크립트를 사용하여 **openstack-tripleo-heat-templates** 컬렉션 사본을 렌더링합니다.

```
$ cd /usr/share/openstack-tripleo-heat-templates
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

이렇게 하면 모든 **Jinja2** 템플릿을 렌더링된 **YAML** 버전으로 변환하고 결과를 **~/openstack-tripleo-heat-templates-rendered**에 저장합니다.

사용자 지정 네트워크 파일 또는 사용자 지정 역할 파일을 사용하는 경우 각각 **-n** 및 **- r** 옵션을 사용하여 이러한 파일을 포함할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered -n
/home/stack/network_data.yaml -r /home/stack/roles_data.yaml
```

2. 여러 **NIC** 예제를 복사합니다.

```
$ cp -r ~/openstack-tripleo-heat-templates-rendered/network/config/multiple-nics/
~/templates/custom-nics/
```

3. **custom-nics** 에 설정된 템플릿을 편집하여 자체 네트워크 구성에 맞게 편집할 수 있습니다.

11.3. 네트워크 인터페이스 아키텍처

11.2절. "사용자 정의를 위한 기본 네트워크 인터페이스 템플릿 렌더링" 에서 렌더링하는 사용자 정의 **NIC** 템플릿에는 매개변수 및 리소스 섹션이 포함되어 있습니다.

매개 변수

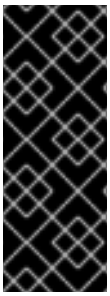
parameters 섹션에는 네트워크 인터페이스에 대한 모든 네트워크 구성 매개 변수가 포함되어 있습니다. 여기에는 서브넷 범위 및 **VLAN ID**와 같은 정보가 포함됩니다. **Heat** 템플릿이 상위 템플릿에서 값을 상속하므로 이 섹션은 변경되지 않은 상태로 유지됩니다. 그러나 네트워크 환경 파일을 사용하여 일부 매개변수의 값을 수정할 수 있습니다.

Resources

resources 섹션은 기본 네트워크 인터페이스 구성이 발생하는 위치입니다. 대부분의 경우 **resources** 섹션은 편집이 필요한 유일한 섹션입니다. 각 **resources** 섹션은 다음 헤더로 시작합니다.

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
```

이렇게 하면 노드에서 네트워크 속성을 구성하는 데 사용할 **os-net-config**의 구성 파일을 생성하는 스크립트(**run-os-net-config.sh**)가 실행됩니다. **network_config** 섹션에는 **run-os-net-config.sh** 스크립트로 전송된 사용자 지정 네트워크 인터페이스 데이터가 포함되어 있습니다. 이 사용자 지정 인터페이스 데이터를 장치 유형에 따라 시퀀스로 정렬합니다.



중요

사용자 지정 **NIC** 템플릿을 생성하는 경우 **run-os-net-config.sh** 스크립트 위치를 각 **NIC** 템플릿의 절대 위치로 설정해야 합니다. 이 스크립트는 언더클라우드의 **/usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh** 에 있습니다.

11.4. 네트워크 인터페이스 참조

네트워크 인터페이스 구성에는 다음 매개변수가 포함됩니다.

인터페이스

단일 네트워크 인터페이스를 정의합니다. 구성은 실제 인터페이스 이름(**"eth0", "eth1", "enp0s25"**) 또는 번호가 매겨진 인터페이스 집합(**"nic1", "nic2", "nic3"**)을 사용하여 각 인터페이스를 정의합니다.

예를 들면 다음과 같습니다.

```
- type: interface
  name: nic2
```

표 11.1. 인터페이스 옵션

옵션	Default	설명
name		인터페이스 이름
use_dhcp	False	DHCP를 사용하여 IP 주소 가져오기
use_dhcpv6	False	DHCP를 사용하여 v6 IP 주소 가져오기
주소		인터페이스에 할당된 IP 주소 목록
routes		인터페이스에 할당된 경로 목록입니다. routes을 참조하십시오.
mtu	1500	연결의 최대 전송 단위 (MTU)
주	False	인터페이스를 기본 인터페이스로 정의합니다.
defroute	True	DHCP 서비스에서 제공하는 기본 경로를 사용합니다. **use_dhcp 또는 use_dhcp v6** 이 활성화된 경우에만 적용됩니다.
persist_mapping	False	시스템 이름이 아닌 장치 별칭 구성 쓰기
dhclient_args	없음	DHCP 클라이언트로 전달할 인수
dns_servers	없음	인터페이스에 사용할 DNS 서버 목록
ethtool_opts		특정 NIC에서 VXLAN을 사용할 때 처리량을 개선하기 위해 이 옵션을 **"rx-flow-hash udp4 sdfn"** 으로 설정합니다.

VLAN

VLAN을 정의합니다. parameters 섹션에서 전달된 VLAN ID 및 서브넷을 사용합니다.

예를 들면 다음과 같습니다.

```
- type: vlan
  vlan_id:{get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
```

표 11.2. VLAN 옵션

옵션	Default	설명
vlan_id		VLAN ID
장치		VLAN을 연결할 상위 장치입니다. VLAN이 OVS 브리지의 멤버가 아닌 경우 이 매개 변수를 사용합니다. 예를 들어 이 매개 변수를 사용하여 VLAN을 본딩된 인터페이스 장치에 연결합니다.
use_dhcp	False	DHCP를 사용하여 IP 주소 가져오기
use_dhcpv6	False	DHCP를 사용하여 v6 IP 주소 가져오기
주소		VLAN에 할당된 IP 주소 목록
routes		VLAN에 할당된 경로 목록입니다. routes을 참조하십시오.
mtu	1500	연결의 최대 전송 단위 (MTU)
주	False	VLAN을 기본 인터페이스로 정의합니다.
defroute	True	DHCP 서비스에서 제공하는 기본 경로를 사용합니다. **use_dhcp 또는 use_dhcp v6** 이 활성화된 경우에만 적용됩니다.
persist_mapping	False	시스템 이름이 아닌 장치 별칭 구성 쓰기
dhclient_args	없음	DHCP 클라이언트로 전달할 인수

옵션	Default	설명
dns_servers	없음	VLAN에 사용할 DNS 서버 목록

ovs_bond

두 개 이상의 인터페이스를 함께 결합할 **Open vSwitch**의 본딩을 정의합니다. 이는 이중화 및 대역폭 증가에 도움이 됩니다.

예를 들면 다음과 같습니다.

```
- type: ovs_bond
  name: bond1
  members:
  - type: interface
    name: nic2
  - type: interface
    name: nic3
```

표 11.3. ovs_bond 옵션

옵션	Default	설명
name		본딩 이름
use_dhcp	False	DHCP를 사용하여 IP 주소 가져오기
use_dhcpv6	False	DHCP를 사용하여 v6 IP 주소 가져오기
주소		본딩에 할당된 IP 주소 목록
routes		본딩에 할당된 경로 목록입니다. routes을 참조하십시오.
mtu	1500	연결의 최대 전송 단위 (MTU)
주	False	인터페이스를 기본 인터페이스로 정의합니다.
멤버		본딩에서 사용할 인터페이스 오브젝트 시퀀스
ovs_options		본딩을 만들 때 OVS에 전달할 옵션 세트

옵션	Default	설명
ovs_extra		본딩의 네트워크 구성 파일에서 OVS_EXTRA 매개 변수로 설정할 옵션 세트
defroute	True	DHCP 서비스에서 제공하는 기본 경로를 사용합니다. **use_dhcp 또는 use_dhcp v6** 이 활성화된 경우에만 적용됩니다.
persist_mapping	False	시스템 이름이 아닌 장치 별칭 구성 쓰기
dhclient_args	없음	DHCP 클라이언트로 전달할 인수
dns_servers	없음	본딩에 사용할 DNS 서버 목록

ovs_bridge

여러 인터페이스,ovs_bond 및 vlan 오브젝트를 함께 연결하는 Open vSwitch에 브리지를 정의합니다.

네트워크 인터페이스 유형인 ovs_bridge 는 매개 변수 이름을 사용합니다.

참고

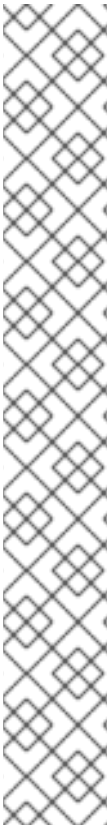
여러 브리지가 있는 경우 bridge _name의 기본 이름을 허용하는 것 이외의 고유한 브리지 이름을 사용해야 합니다. 고유한 이름을 사용하지 않으면 통합 단계에서 두 개의 네트워크 본딩이 동일한 브리지에 배치됩니다.

외부 tripleo 네트워크에 대한 OVS 브리지를 정의하는 경우 배포 프레임워크에서 이러한 값을 각각 외부 브리지 이름 및 외부 인터페이스 이름으로 교체하므로 bridge_name 및 interface_name 값을 유지합니다.

예를 들면 다음과 같습니다.

```
- type: ovs_bridge
  name: bridge_name
  addresses:
  - ip_netmask:
      list_join:
```

```
        - /
        - - {get_param: ControlPlaneIp}
          - {get_param: ControlPlaneSubnetCidr}
      members:
        - type: interface
          name: interface_name
    - type: vlan
      device: bridge_name
      vlan_id:
        {get_param: ExternalNetworkVlanID}
      addresses:
        - ip_netmask:
            {get_param: ExternalIpSubnet}
```



참고

OVS 브리지는 구성 데이터를 가져오기 위해 Neutron 서버에 연결합니다. OpenStack 제어 트래픽(일반적으로 컨트롤 플레인 및 내부 API 네트워크)이 OVS 브리지에 배치되면 OVS가 업그레이드되거나 관리자 또는 프로세스에서 OVS 브리지를 다시 시작할 때마다 Neutron 서버에 대한 연결이 손실됩니다. 이로 인해 일부 다운타임이 발생합니다. 이러한 상황에서 다운타임을 허용하지 않는 경우, 제어 그룹 네트워크를 OVS 브리지가 아닌 별도의 인터페이스 또는 본딩에 배치해야 합니다.

- **프로비저닝 인터페이스의 VLAN에 내부 API 네트워크를 배치하고 두 번째 인터페이스의 OVS 브리지에 내부 API 네트워크를 배치하면 최소 설정을 달성할 수 있습니다.**

- **본딩을 사용하려면 2개 이상의 본딩(4개의 네트워크 인터페이스)이 필요합니다. 제어 그룹은 Linux 본딩(Linux 브리지)에 배치되어야 합니다. 스위치에서 PXE 부팅을 위한 단일 인터페이스로 LACP 폴백을 지원하지 않는 경우 이 솔루션에는 최소 5개의 NIC가 필요합니다.**

표 11.4. ovs_bridge options

옵션	Default	설명
name		브리지 이름
use_dhcp	False	DHCP를 사용하여 IP 주소 가져오기
use_dhcpv6	False	DHCP를 사용하여 v6 IP 주소 가져오기
주소		브리지에 할당된 IP 주소 목록

옵션	Default	설명
routes		브리지에 할당된 경로 목록입니다. routes을 참조하십시오.
mtu	1500	연결의 최대 전송 단위 (MTU)
멤버		브리지에서 사용할 인터페이스, VLAN, 본딩 오브젝트의 시퀀스
ovs_options		브리지를 만들 때 OVS에 전달할 옵션 세트
ovs_extra		브리지 네트워크 구성 파일에서 OVS_EXTRA 매개 변수로 설정할 옵션 세트
defroute	True	DHCP 서비스에서 제공하는 기본 경로를 사용합니다. **use_dhcp 또는 use_dhcp v6** 이 활성화된 경우에만 적용됩니다.
persist_mapping	False	시스템 이름이 아닌 장치 별칭 구성 쓰기
dhclient_args	없음	DHCP 클라이언트로 전달할 인수
dns_servers	없음	브리지에 사용할 DNS 서버 목록

linux_bond

두 개 이상의 인터페이스를 함께 결합하는 Linux 본딩을 정의합니다. 이는 이중화 및 대역폭 증가에 도움이 됩니다. bonding _options 매개 변수에 커널 기반 본딩 옵션을 포함시켜야 합니다. Linux 본딩 옵션에 대한 자세한 내용은 7.7.1을 참조하십시오. Red Hat Enterprise Linux 7 네트워킹 가이드의 본딩 모듈 지시문

예를 들면 다음과 같습니다.

```
- type: linux_bond
  name: bond1
  members:
  - type: interface
    name: nic2
    primary: true
```

```
    - type: interface
      name: nic3
    bonding_options: "mode=802.3ad"
```

nic2 에서는 primary: true 를 사용합니다. 이렇게 하면 본딩에서 nic2 의 MAC 주소를 사용합니다.

표 11.5. linux_bond 옵션

옵션	Default	설명
name		본딩 이름
use_dhcp	False	DHCP를 사용하여 IP 주소 가져오기
use_dhcpv6	False	DHCP를 사용하여 v6 IP 주소 가져오기
주소		본딩에 할당된 IP 주소 목록
routes		본딩에 할당된 경로 목록입니다. routes을 참조하십시오.
mtu	1500	연결의 최대 전송 단위 (MTU)
주	False	인터페이스를 기본 인터페이스로 정의합니다.
멤버		본딩에서 사용할 인터페이스 오브젝트 시퀀스
bonding_options		본딩 생성 시 옵션 집합입니다. Linux 본딩 옵션에 대한 자세한 내용은 7.7.1을 참조하십시오. Red Hat Enterprise Linux 7 네트워킹 가이드의 본딩 모듈 지시문
defroute	True	DHCP 서비스에서 제공하는 기본 경로를 사용합니다. **use_dhcp 또는 use_dhcp v6** 이 활성화된 경우에만 적용됩니다.
persist_mapping	False	시스템 이름이 아닌 장치 별칭 구성 쓰기
dhclient_args	없음	DHCP 클라이언트로 전달할 인수
dns_servers	없음	본딩에 사용할 DNS 서버 목록

linux_bridge

여러 인터페이스,linux_bond 및 vlan 오브젝트를 함께 연결하는 Linux 브리지를 정의합니다. 외부 브리지는 매개 변수에 두 개의 특수 값도 사용합니다.

- **bridge_name - 외부 브리지 이름으로 바뀝니다.**
- **interface_name - 외부 인터페이스로 바뀝니다.**

예를 들면 다음과 같습니다.

```
- type: linux_bridge
  name: bridge_name
  addresses:
    - ip_netmask:
        list_join:
          - /
          - - {get_param: ControlPlaneIp}
            - {get_param: ControlPlaneSubnetCidr}
  members:
    - type: interface
      name: interface_name
- type: vlan
  device: bridge_name
  vlan_id:
    {get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask:
        {get_param: ExternalIpSubnet}
```

표 11.6. linux_bridge options

옵션	Default	설명
name		브리지 이름
use_dhcp	False	DHCP를 사용하여 IP 주소 가져오기
use_dhcpv6	False	DHCP를 사용하여 v6 IP 주소 가져오기
주소		브리지에 할당된 IP 주소 목록
routes		브리지에 할당된 경로 목록입니다. routes을 참조하십시오.

옵션	Default	설명
mtu	1500	연결의 최대 전송 단위 (MTU)
멤버		브리지에서 사용할 인터페이스, VLAN, 본딩 오브젝트의 시퀀스
defroute	True	DHCP 서비스에서 제공하는 기본 경로를 사용합니다. **use_dhcp 또는 use_dhcp v6** 이 활성화된 경우에만 적용됩니다.
persist_mapping	False	시스템 이름이 아닌 장치 별칭 구성 쓰기
dhclient_args	없음	DHCP 클라이언트로 전달할 인수
dns_servers	없음	브리지에 사용할 DNS 서버 목록

routes

네트워크 인터페이스, VLAN, 브리지 또는 본딩에 적용할 경로 목록을 정의합니다.

예를 들면 다음과 같습니다.

```
- type: interface
  name: nic2
  ...
  routes:
    - ip_netmask: 10.1.2.0/24
      default: true
      next_hop:
        get_param: EC2MetadataIp
```

옵션	Default	설명
ip_netmask	없음	대상 네트워크의 IP 및 넷마스크.
default	False	이 경로를 기본 경로로 설정합니다. **ip_netmask 설정과 같습니다. 0.0.0.0/0**.

옵션	Default	설명
next_hop	없음	대상 네트워크에 연결하는 데 사용되는 라우터의 IP 주소입니다.

11.5. 네트워크 인터페이스 레이아웃 예

컨트롤러 노드 NIC 템플릿 예제의 다음 코드 조각은 OVS 브리지와 별도로 제어 그룹을 유지하도록 사용자 지정 네트워크 시나리오를 구성하는 방법을 보여줍니다.

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:

              # NIC 1 - Provisioning
              - type: interface
                name: nic1
                use_dhcp: false
                addresses:
                - ip_netmask:
                    list_join:
                    - /
                    - - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
                routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop:
                    get_param: EC2MetadataIp

              # NIC 2 - Control Group
              - type: interface
                name: nic2
                use_dhcp: false
              - type: vlan
                device: nic2
                vlan_id:
                  get_param: InternalApiNetworkVlanID
                addresses:
                - ip_netmask:
                    get_param: InternalApiIpSubnet
              - type: vlan
```

```
    device: nic2
    vlan_id:
      get_param: StorageMgmtNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: StorageMgmtIpSubnet
  - type: vlan
    device: nic2
    vlan_id:
      get_param: ExternalNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
    routes:
    - default: true
      next_hop:
        get_param: ExternalInterfaceDefaultRoute

  # NIC 3 - Data Group
  - type: ovs_bridge
    name: bridge_name
    dns_servers:
      get_param: DnsServers
    members:
    - type: interface
      name: nic3
      primary: true
    - type: vlan
      vlan_id:
        get_param: StorageNetworkVlanID
      addresses:
      - ip_netmask:
          get_param: StorageIpSubnet
    - type: vlan
      vlan_id:
        get_param: TenantNetworkVlanID
      addresses:
      - ip_netmask:
          get_param: TenantIpSubnet

    # NIC 4 - Management
    - type: interface
      name: nic4
      use_dhcp: false
      addresses:
      - ip_netmask: {get_param: ManagementIpSubnet}
      routes:
      - default: true
        next_hop: {get_param: ManagementInterfaceDefaultRoute}
```

이 템플릿은 4개의 네트워크 인터페이스를 사용하고 번호가 매겨진 인터페이스인 nic1을 nic 4 에 할당합니다. nic3 에서 스토리지 및 테넌트 네트워크를 호스팅하는 OVS 브리지를 만듭니다. 결과적으로 다음과 같은 레이아웃이 생성됩니다.

- NIC1 (프로비저닝):
  - 프로비저닝/컨트롤 플레인
- NIC2 (Control Group)
  - 내부 API
  - 스토리지 관리
  - 외부(공용 API)
- NIC3 (데이터 그룹)
  - 테넌트 네트워크(VXLAN 터널링)
  - 테넌트 VLAN/프로바이더 VLAN
  - 스토리지
  - 외부 VLAN (유동 IP/SNAT)
- NIC4 (관리)
  - 관리

11.6. 사용자 지정 네트워크의 네트워크 인터페이스 템플릿 고려 사항

구성 가능 네트워크를 사용하는 경우 process-templates.py 스크립트는 정적 템플릿을 렌더링하여 network_data.yaml 및 roles _data. yaml 파일에 정의된 네트워크 및 역할을 포함합니다. 렌더링된 NIC

템플릿에 다음 항목이 포함되어 있는지 확인합니다.

- **사용자 지정 구성 가능 네트워크를 포함하여 각 역할에 대한 정적 파일입니다.**
- **각 역할에 대한 각 정적 파일에는 올바른 네트워크 정의가 포함되어 있습니다.**

각 정적 파일에는 네트워크를 역할에 사용하지 않아도 사용자 지정 네트워크의 모든 매개 변수 정의가 필요합니다. 렌더링된 템플릿에 이러한 매개 변수가 포함되어 있는지 확인합니다. 예를 들어 StorageBackup 네트워크가 Ceph 노드에만 추가되면 모든 역할에 대한 NIC 구성 템플릿의 parameters 섹션에 이 정의도 포함되어야 합니다.

```
parameters:
  ...
  StorageBackupIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ...
```

필요한 경우 VLAN ID 및/또는 게이트웨이 IP에 대한 매개변수 정의를 포함할 수도 있습니다.

```
parameters:
  ...
  StorageBackupNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  StorageBackupDefaultRoute:
   description: The default route of the storage backup network.
   type: string
  ...
```

사용자 지정 네트워크의 IpSubnet 매개변수는 각 역할의 매개 변수 정의에 표시됩니다. 그러나 Ceph 역할이 StorageBackup 네트워크를 사용하는 유일한 역할일 수 있으므로 Ceph 역할의 NIC 구성 템플릿만 템플릿의 network_config 섹션에 있는 StorageBackup 매개 변수를 사용합니다.

```
    $network_config:
      network_config:
      - type: interface
        name: nic1
        use_dhcp: false
        addresses:
        - ip_netmask:
            get_param: StorageBackupIpSubnet
```

11.7. 사용자 지정 네트워크 환경 파일

사용자 지정 네트워크 환경 파일(이 경우 **/home/stack/templates/custom-network-configuration.yaml**)은 **Overcloud** 네트워크 환경을 설명하고 사용자 지정 네트워크 인터페이스 구성 템플릿을 가리키는 **heat** 환경 파일입니다. **IP** 주소 범위와 함께 네트워크의 서브넷 및 **VLAN**을 정의할 수 있습니다. 그런 다음 로컬 환경에 대해 이러한 값을 사용자 지정할 수 있습니다.

resource_registry 섹션에는 각 노드 역할에 대한 사용자 지정 네트워크 인터페이스 템플릿에 대한 참조가 포함되어 있습니다. 등록된 각 리소스는 다음 형식을 사용합니다.

- **OS::TripleO::[ROLE]::Net::SoftwareConfig: [FILE]**

[ROLE] 은 역할 이름이고 **[FILE]** 은 해당 특정 역할에 대한 각 네트워크 인터페이스 템플릿입니다. 예를 들면 다음과 같습니다.

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/custom-nics/controller.yaml
```

parameter_defaults 섹션에는 각 네트워크 유형에 대한 네트워크 옵션을 정의하는 매개 변수 목록이 포함되어 있습니다.

11.8. 네트워크 환경 매개변수

다음 표는 네트워크 환경 파일의 **parameter_defaults** 섹션에서 **NIC** 템플릿의 기본 매개변수 값을 재정의하는 데 사용할 수 있는 매개변수 목록입니다.

매개변수	설명	유형
ControlPlaneDefaultRoute	기본적으로 컨트롤러 노드 이외의 역할에 대한 기본 경로로 사용되는 컨트롤 플레인의 라우터 IP 주소입니다. 라우터 대신 IP masquerade를 사용하는 경우 Undercloud IP로 설정합니다.	string
ControlPlaneSubnetCidr	컨트롤 플레인에서 사용되는 IP 네트워크의 CIDR 넷마스크입니다. 컨트롤 플레인 네트워크에서 192.168.24.0/24를 사용하는 경우 CIDR은 **24** 입니다.	문자열 (항상 숫자임)

매개변수	설명	유형
***NetCidr**	특정 네트워크의 전체 네트워크 및 CIDR 넷마스크. 기본값은 network_**data** 파일에서 네트워크의 **ip_subnet** 설정으로 자동으로 설정됩니다. 예를 들면 다음과 같습니다. **InternalApiNetCidr: 172.16.0.0/24**	string
***AllocationPools**	"특정 네트워크에 대한 IP 할당 범위. 기본값은 network_**data** 파일에서 네트워크의 **allocation_pools** 설정으로 자동 설정됩니다. 예를 들면 다음과 같습니다. **InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]**	해시
***NetworkVlanID**	특정 네트워크의 에 대한 노드의 VLAN ID입니다. 기본값은 network**_data** 파일에서 네트워크의 **vlan** 설정으로 자동 설정됩니다. 예를 들면 다음과 같습니다. **InternalApiNetworkVlanID: 201**	숫자
***InterfaceDefaultRoute**	역할의 기본 경로로 사용하거나 다른 네트워크의 경로에 사용할 수 있는 특정 네트워크의 라우터 주소입니다. 기본값은 network_**data** 파일에서 네트워크의 **gateway_ip** 설정으로 자동으로 설정됩니다. 예를 들면 다음과 같습니다. **InternalApiInterfaceDefaultRoute: 172.16.0.1**	string
DnsServers	resolv.conf에 추가된 DNS 서버 목록입니다. 일반적으로 최대 2대의 서버를 허용합니다.	쉼표로 구분된 목록
EC2MetadataIp	Overcloud 노드를 프로비저닝하는 데 사용되는 메타데이터 서버의 IP 주소입니다. 컨트롤 플레인에서 언더클라우드의 IP 주소로 설정합니다.	string

매개변수	설명	유형
BondInterfaceOvsOptions	본딩 인터페이스에 대한 옵션. 예를 들면 다음과 같습니다. **BondInterfaceOvsOptions: "bond_mode=balance-slb"**	string
NeutronExternalNetworkBridge	OpenStack Networking(neutron)에 사용할 외부 브리지 이름의 레거시 값입니다. 이 값은 기본적으로 비어 있으며 **NeutronBridgeMappings** 에 여러 물리 브리지를 정의할 수 있습니다. 일반적으로 이 값을 재정의해서는 안 됩니다.	string
NeutronFlatNetworks	neutron 플러그인에 구성할 플랫 네트워크를 정의합니다. 외부 네트워크 생성을 허용하려면 기본값은 "datacentre"입니다. 예를 들면 다음과 같습니다. **NeutronFlatNetworks: "datacentre"**	string
NeutronBridgeMappings	사용할 논리적 브릿지와 물리적 브리지 매핑. 기본적으로 호스트(**br-ex**)의 외부 브리지를 물리적 이름(datacentre)에 매핑**합니다**. OpenStack Networking(neutron) 프로바이더 네트워크 또는 유동 IP 네트워크를 만들 때 논리 이름을 참조합니다. 예를 들어 **NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"**	string
NeutronPublicInterface	네트워크 격리를 사용하지 않는 경우 네트워크 노드의 **br-ex** 에 브리지에 인터페이스를 정의합니다. 일반적으로 두 개의 네트워크만 있는 소규모 배포에서는 사용되지 않습니다. 예를 들면 다음과 같습니다. **NeutronPublicInterface: "eth0"**	string

매개변수	설명	유형
NeutronNetworkType	t: OpenStack Networking(neutron)의 테넌트 네트워크 유형. 여러 값을 지정하려면 쉼표로 구분된 목록을 사용합니다. 지정된 첫 번째 유형은 사용 가능한 모든 네트워크가 모두 사용될 때까지 사용되고 다음 유형이 사용됩니다. 예를 들면 다음과 같습니다. **NeutronNetworkType: "vxlan"**	string
NeutronTunnelTypes	neutron 테넌트 네트워크의 터널 유형입니다. 여러 값을 지정하려면 쉼표로 구분된 문자열을 사용합니다. 예를 들면 다음과 같습니다. *NeutronTunnelTypes: 'gre,vxlan'*	문자열 / 쉼표로 구분된 목록
NeutronTunnelIdRanges	테넌트 네트워크 할당에 사용할 수 있는 GRE 터널 ID 범위입니다. 예를 들면 다음과 같습니다. **NeutronTunnelIdRanges "1:1000"**	string
NeutronVniRanges	테넌트 네트워크 할당에 사용할 수 있는 VXLAN VNI ID 범위입니다. 예를 들면 다음과 같습니다. **NeutronVniRanges: "1:1000"**	string
NeutronEnableTunnelling	터널링된 모든 네트워크를 활성화 또는 완전히 비활성화할지 여부를 정의합니다. 튜닝된 네트워크를 생성하지 않으려면 이 기능을 활성화하십시오. 기본값은 enabled입니다.	부울
NeutronNetworkVLANRanges	지원할 ML2 및 Open vSwitch VLAN 매핑 범위. 기본값은 **datacentre** 물리적 네트워크에서 VLAN을 허용하도록 합니다. 여러 값을 지정하려면 쉼표로 구분된 목록을 사용합니다. 예를 들면 다음과 같습니다. **NeutronNetworkVLANRanges: "datacentre:1:1000,tenant:100:299,tenant:310:399"**	string

매개변수	설명	유형
NeutronMechanismDrivers	neutron 테넌트 네트워크의 메커니즘 드라이버입니다. 기본값은 "openvswitch"입니다. 여러 값을 지정하려면 쉼표로 구분된 문자열을 사용합니다. 예를 들면 다음과 같습니다. **NeutronMechanismDrivers: 'openvswitch,l2population'**	문자열 / 쉼표로 구분된 목록

11.9. 사용자 지정 네트워크 환경 파일 예

다음 코드 조각은 **NIC** 템플릿을 활성화하고 사용자 정의 매개변수를 설정하는 데 사용할 수 있는 환경 파일의 예입니다.

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

parameter_defaults:
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.2.254
  # The IP address of the EC2 metadata server. Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.2.1
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["8.8.8.8","8.8.4.4"]
  NeutronExternalNetworkBridge: """
```

11.10. 사용자 정의 NIC를 통한 네트워크 격리 활성화

네트워크 분리 및 사용자 지정 **NIC** 템플릿을 사용하여 **Overcloud**를 배포하려면 오버클라우드 배포 명령의 모든 관련 네트워킹 환경 파일을 포함합니다.

절차

1. **openstack overcloud deploy** 명령을 실행할 때 다음을 포함해야 합니다.

- 사용자 지정 **network_data** 파일.
- 기본 네트워크 격리의 렌더링된 파일 이름입니다.
- 렌더링된 기본 네트워크 환경 파일의 파일 이름입니다.
- 사용자 지정 **NIC** 템플릿에 대한 리소스 참조가 포함된 사용자 지정 환경 네트워크 구성입니다.
- 구성과 관련된 추가 환경 파일입니다.

예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  ...
  -n /home/stack/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /home/stack/templates/custom-network-configuration.yaml \
  ...
```

- 먼저 **network-isolation.yaml** 파일을 포함하고 **network-environment.yaml** 파일을 포함합니다. 후속 **custom-network-configuration.yaml** 은 이전 두 파일의 **OS::TripleO::[ROLE]::Net::SoftwareConfig** 리소스를 덮어씁니다.
- 구성 가능한 네트워크를 사용하는 경우 이 명령으로 **network_data** 및 **roles_data** 파일을 포함합니다.

12장. 추가 네트워크 구성

이 장에서는 11장. *사용자 정의 네트워크 인터페이스 템플릿* 에 설명된 개념 및 절차에서 설명하고 오버클라우드 네트워크의 일부를 구성하는 데 도움이 되는 몇 가지 추가 정보를 제공합니다.

12.1. 사용자 정의 인터페이스 구성

개별 인터페이스에는 수정이 필요할 수 있습니다. 다음 예제에서는 두 번째 NIC를 사용하여 DHCP 주소가 있는 인프라 네트워크에 연결하고 본딩에 세 번째 및 네 번째 NIC를 사용하는 데 필요한 수정 사항을 보여줍니다.

```
network_config:
  # Add a DHCP infrastructure network to nic2
  - type: interface
    name: nic2
    use_dhcp: true
  - type: ovs_bridge
    name: br-bond
    members:
      - type: ovs_bond
        name: bond1
        ovs_options:
          get_param: BondInterfaceOvsOptions
        members:
          # Modify bond NICs to use nic3 and nic4
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```

네트워크 인터페이스 템플릿에서는 실제 인터페이스 이름(eth0, eth 1,enp0s25) 또는번호가 매겨진 인터페이스 집합(nic1, nic 2, nic 3)을 사용합니다. 역할 내의 호스트의 네트워크 인터페이스는 명명된 인터페이스(eth0,eno2 등) 대신 번호가 지정된 인터페이스(nic1,nic2 등)를 사용할 때 정확하게 동일할 필요는 없습니다. 예를 들어 한 호스트에는 em1 및 em 2 인터페이스가 있을 수 있지만 다른 호스트는 eno1 및 eno2 가 있지만 두 호스트의 NIC를 모두 nic1 및 nic 2 로 참조할 수 있습니다.

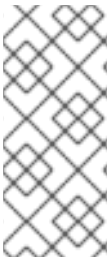
번호가 매겨진 인터페이스 순서는 명명된 네트워크 인터페이스 유형의 순서에 해당합니다.

- eth 0, eth 1 등과 같은 eth X 인터페이스. 일반적으로 온보드 인터페이스입니다.
- enoX 인터페이스(예: eno0,eno1) 등. 일반적으로 온보드 인터페이스입니다.

- **enX** 인터페이스, **enp3s 0, enp3s 1,ens3** , 등과 같은 영숫자로 정렬됩니다. 일반적으로 애드온 인터페이스입니다.

번호가 매겨진 **NIC** 스키마는 스위치에 연결된 케이블이 있는 경우 라이브 인터페이스(예:)만 고려합니다. 인터페이스가 **4**개이고 인터페이스가 **6**개인 일부 호스트가 있는 경우 **nic1**을 **nic 4** 에 사용하고 각 호스트에 **4**개의 케이블만 연결해야 합니다.

물리적 인터페이스를 특정 별칭으로 하드 코딩할 수 있습니다. 이를 통해 **nic1** 또는 **nic 2** 등으로 매핑될 물리적 **NIC**를 미리 결정할 수 있습니다. **MAC** 주소를 지정된 별칭에 매핑할 수도 있습니다.



참고

일반적으로 **os-net-config** 는 **UP** 상태로 이미 연결된 인터페이스만 등록합니다. 그러나 사용자 지정 매핑 파일을 사용하여 인터페이스를 하드 코딩하는 경우 **DOWN** 상태인 경우에도 인터페이스가 등록됩니다.

인터페이스는 환경 파일을 사용하여 별칭에 매핑됩니다. 이 예에서 각 노드에는 **nic1** 및 **nic 2** 에 대한 사전 정의된 항목이 있습니다.

참고

NetConfigDataLookup 구성을 사용하려면 **NodeUserData** 리소스 레지스트리에 **os-net-config-mappings.yaml** 파일도 포함해야 합니다.

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack/tripleo-heat-templates/firstboot/os-net-config-
mappings.yaml
parameter_defaults:
  NetConfigDataLookup:
    node1:
      nic1: "em1"
      nic2: "em2"
    node2:
      nic1: "00:50:56:2F:9F:2E"
      nic2: "em2"
```

그런 다음 결과 구성이 **os-net-config** 에 의해 적용됩니다. 각 노드에서 **/etc/os-net-config/mapping.yaml** 파일의 **interface_mapping** 섹션에서 적용된 구성을 확인할 수 있습니다.

12.2. 경로 및 기본 경로 구성

두 가지 방법 중 하나로 호스트의 기본 경로를 설정할 수 있습니다. 인터페이스에서 **DHCP**를 사용하고 **DHCP** 서버에서 게이트웨이 주소를 제공하는 경우 시스템은 해당 게이트웨이의 기본 경로를 사용합니다. 그렇지 않으면 정적 **IP**를 사용하여 인터페이스에서 기본 경로를 설정할 수 있습니다.

Linux 커널은 여러 기본 게이트웨이를 지원하지만 가장 낮은 메트릭이 있는 게이트웨이만 사용합니다. **DHCP** 인터페이스가 여러 개인 경우 예측할 수 없는 기본 게이트웨이가 발생할 수 있습니다. 이 경우 기본 경로를 사용하는 인터페이스 이외의 인터페이스에 대해 **defroute: false** 를 설정하는 것이 좋습니다.

예를 들어 **DHCP** 인터페이스(**nic3**)를 기본 경로로지정할 수 있습니다. 다음 **YAML**을 사용하여 다른 **DHCP** 인터페이스(**nic2**)에서 기본 경로를 비활성화합니다.

```
# No default route on this DHCP interface
- type: interface
  name: nic2
  use_dhcp: true
  defroute: false
# Instead use this DHCP interface as the default route
- type: interface
  name: nic3
  use_dhcp: true
```

참고

defroute 매개 변수는 **DHCP**를 통해 얻은 경로에만 적용됩니다.

고정 **IP**가 있는 인터페이스에서 고정 경로를 설정하려면 서브넷의 경로를 지정합니다. 예를 들어 내부 **API** 네트워크의 **172.17.0.1**의 게이트웨이를 통해 **10.1.2.0/24** 서브넷으로 경로를 설정할 수 있습니다.

```
- type: vlan
  device: bond1
  vlan_id:
    get_param: InternalApiNetworkVlanID
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
  - ip_netmask: 10.1.2.0/24
    next_hop: 172.17.0.1
```

12.3. 점보 프레임 구성

MTU(최대 전송 단위) 설정은 단일 이더넷 프레임으로 전송되는 최대 데이터 양을 결정합니다. 더 큰 값을 사용하면 각 프레임이 헤더 형태로 데이터가 추가되기 때문에 오버헤드가 줄어듭니다. 기본값은 1500이며 더 높은 값을 사용하려면 점보 프레임을 지원하도록 스위치 포트를 구성해야 합니다. 대부분의 스위치는 9000 MTU를 지원하지만, 기본적으로 1500용으로 구성되어 있습니다.

VLAN의 MTU는 실제 인터페이스의 MTU를 초과할 수 없습니다. 본딩 및/또는 인터페이스에 MTU 값을 포함해야 합니다.

스토리지, 스토리지 관리, 내부 API 및 테넌트 네트워크는 모두 점보 프레임의 이점을 제공합니다. 테스트에서 프로젝트의 네트워킹 처리량은 VXLAN 터널과 함께 점보 프레임을 사용할 때 상당히 개선되었습니다.

참고

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

```
- type: ovs_bond
  name: bond1
  mtu: 9000
  ovs_options: {get_param: BondInterfaceOvsOptions}
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000

# The external interface should stay at default
- type: vlan
  device: bond1
  vlan_id:
    get_param: ExternalNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
  routes:
    - ip_netmask: 0.0.0.0/0
      next_hop:
        get_param: ExternalInterfaceDefaultRoute

# MTU 9000 for Internal API, Storage, and Storage Management
- type: vlan
  device: bond1
```

```
  mtu: 9000
  vlan_id:
    get_param: InternalApiNetworkVlanID
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
```

12.4. 트렁크된 인터페이스에서 기본 VLAN 구성

트렁크된 인터페이스 또는 본딩에 기본 **VLAN**에 네트워크가 있는 경우 **IP** 주소가 브리지에 직접 할당되고 **VLAN** 인터페이스가 없습니다.

예를 들어 외부 네트워크가 기본 **VLAN**에 있는 경우 본딩된 구성은 다음과 같습니다.

```
network_config:
  - type: ovs_bridge
    name: bridge_name
    dns_servers:
      get_param: DnsServers
    addresses:
      - ip_netmask:
          get_param: ExternalIpSubnet
    routes:
      - ip_netmask: 0.0.0.0/0
        next_hop:
          get_param: ExternalInterfaceDefaultRoute
    members:
      - type: ovs_bond
        name: bond1
        ovs_options:
          get_param: BondInterfaceOvsOptions
        members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```

참고

주소(및 라우팅) 문을 브리지로 이동할 때 브리지에서 해당 **VLAN** 인터페이스를 제거합니다. 적용 가능한 모든 역할을 변경합니다. 외부 네트워크는 컨트롤러에만 있으므로 컨트롤러 템플릿만 변경해야 합니다. 반면 스토리지 네트워크는 모든 역할에 연결되어 있으므로 스토리지 네트워크가 기본 **VLAN**에 있으면 모든 역할을 수정해야 합니다.

12.5. NETFILTER가 추적하는 최대 연결 수 증가

RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스(neutron)는 netfilter 연결 추적을 사용하여 상태 저장 방화벽을 빌드하고 가상 네트워크에서 NAT(네트워크 주소 변환)를 제공합니다. 커널 공간이 최대 연결 제한에 도달하도록 할 수 있는 상황이 있으며 nf_conntrack: 테이블 전체와 같은 오류가 발생하여 패킷을 삭제할 수 있습니다. 연결 추적(conntrack)의 제한을 늘리고 이러한 유형의 오류를 방지할 수 있습니다. RHOSP 배포에서 하나 이상의 역할 또는 모든 노드에서 conntrack 제한을 늘릴 수 있습니다.

사전 요구 사항

- 성공적인 RHOSP 언더클라우드 설치

절차

1. 언더클라우드 호스트에 stack 사용자로 로그인합니다.

2. 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 사용자 지정 YAML 환경 파일을 생성합니다.

예제

```
$ vi /home/stack/templates/my-environment.yaml
```

4. 환경 파일에는 키워드 parameter_defaults 및 ExtraSysctl 509 가 포함되어야 합니다. netfilter가 변수 net.nf_conntrack_max 에서 추적할 수 있는 최대 연결 수의 새 값을 입력합니다.

예제

이 예제에서는 RHOSP 배포의 모든 호스트에 conntrack 제한을 설정할 수 있습니다.

```
parameter_defaults:
  ExtraSysctlSettings:
```

```
    net.nf_conntrack_max:
      value: 500000
```

<role>Parameter 매개변수를 사용하여 특정 역할에 대한 conntrack 제한을 설정합니다.

```
parameter_defaults:
  <role>Parameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: <simultaneous_connections>
```

- **<role> 을 역할 이름으로 바꿉니다.**

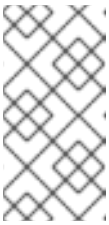
  예를 들어 컨트롤러 매개 변수를 사용하여 컨트롤러 역할에 대한 conntrack 제한을 설정하거나 ComputeParameters 를 사용하여 Compute 역할의 conntrack 제한을 설정합니다.

- **<simultaneous_connections> 를 허용하려는 동시 연결 수로 바꿉니다.**

예제

이 예제에서는 RHOSP 배포의 Controller 역할에 대해서만 conntrack 제한을 설정할 수 있습니다.

```
parameter_defaults:
  ControllerParameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: 500000
```



참고

net.nf_conntrack_max 의 기본값은 500000 연결입니다. 최대값은 다음과 같습니다. 4294967295.

5. **배포 명령을 실행하고 핵심 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.**

중요

후속 환경 파일에 정의된 매개변수 및 리소스가 우선하므로 환경 파일의 순서가 중요합니다.

예제

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/my-environment.yaml
```

추가 리소스

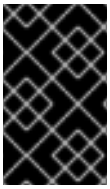
- 환경 파일
- 오버클라우드 생성에 환경 파일 포함

13장. 네트워크 인터페이스 연결

이 장에서는 사용자 지정 네트워크 구성에 사용할 수 있는 몇 가지 본딩 옵션을 정의합니다.

13.1. LACP(NETWORK INTERFACE BONDING AND LINK AGGREGATION CONTROL PROTOCOL)

여러 물리적 NIC를 함께 번들하여 본딩이라는 단일 논리 채널을 구성할 수 있습니다. 고가용성 시스템의 중복성 또는 처리량이 증가하도록 본딩을 구성할 수 있습니다.



중요

결합된 인터페이스에서 SR-IOV(단일 루트 I/O 가상화) 사용은 지원되지 않습니다.

Red Hat OpenStack Platform은 Linux 본딩, OVS(Open vSwitch) 커널 본딩 및 OVS-DPDK 본딩을 지원합니다.

본딩은 선택적 LACP(링크 집계 제어 프로토콜)와 함께 사용할 수 있습니다. LACP는 부하 분산 및 내결함성을 위한 동적 본딩을 생성하는 협상 프로토콜입니다.

Red Hat은 OvS 커널 본딩(bond type: ovs_bond)을 OvS 커널 본딩에 사용하는 것이 좋습니다. 사용자 모드 본딩(결합 유형: ovs_dpdk_bond)은 커널 모드 브리지(type: ovs_user_bridge)와 달리 사용자 모드 브리지(type: ovs_user_bridge)와 함께 사용해야 합니다. 그러나 동일한 노드에서 ovs_bridge 및 ovs_user_bridge를 결합하지 마십시오.

제어 및 스토리지 네트워크에서는 OVS 또는 neutron 에이전트가 업데이트, 핫픽스 및 기타 이벤트를 위해 OVS 또는 neutron 에이전트를 다시 시작할 때 발생할 수 있는 컨트롤 플레인 중단 가능성이 있으므로 Red Hat은 VLAN 및 LACP와 Linux 본딩을 사용할 것을 권장합니다. Linux 본딩/LACP/VLAN 구성은 OVS 중단 가능성 없이 NIC 관리를 제공합니다.

다음은 하나의 VLAN을 사용한 Linux 본딩 구성의 예입니다.

```
params:
        $network_config:
          network_config:

          - type: linux_bond
            name: bond_api
            bonding_options: "mode=active-backup"
```

```
        use_dhcp: false
        dns_servers:
          get_param: DnsServers
        members:
        - type: interface
          name: nic3
          primary: true
        - type: interface
          name: nic4

      - type: vlan
        vlan_id:
          get_param: InternalApiNetworkVlanID
        device: bond_api
        addresses:
        - ip_netmask:
            get_param: InternalApiIpSubnet
```

다음 예는 OVS 브리지에 연결된 Linux 본딩을 보여줍니다.

```
params:
      $network_config:
        network_config:

      -  type: ovs_bridge
         name: br-tenant
         use_dhcp: false
         mtu: 9000
         members:
           - type: linux_bond
             name: bond_tenant
             bonding_options: "mode=802.3ad updelay=1000 miimon=100"
             use_dhcp: false
             dns_servers:
               get_param: DnsServers
             members:
             - type: interface
               name: p1p1
               primary: true
             - type: interface
               name: p1p2
           - type: vlan
             device: bond_tenant
             vlan_id: {get_param: TenantNetworkVlanID}
             addresses:
               -
                 ip_netmask: {get_param: TenantIpSubnet}
```

다음 예는 OVS 사용자 공간 브리지를 보여줍니다.

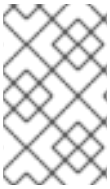
```
params:
```

```
    $network_config:
      network_config:

-     type: ovs_user_bridge
      name: br-ex
      use_dhcp: false
      members:
      - type: ovs_dpdk_bond
        name: dpdkbond0
        mtu: 2140
        ovs_options: {get_param: BondInterfaceOvsOptions}
        #ovs_extra:
        #- set interface dpdk0 mtu_request=$MTU
        #- set interface dpdk1 mtu_request=$MTU
        rx_queue:
          get_param: NumDpdkInterfaceRxQueues
        members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 2140
          members:
          - type: interface
            name: p1p1
        - type: ovs_dpdk_port
          name: dpdk1
          mtu: 2140
          members:
          - type: interface
            name: p1p2
```

13.2. OPEN VSWITCH 본딩 옵션

Overcloud는 OVS(Open vSwitch)를 통해 네트워킹을 제공합니다. 다음 표를 사용하여 결합된 인터페이스의 OVS 커널 및 OVS-DPDK에 대한 지원 호환성을 파악합니다. OVS/OVS-DPDK balance-tcp 모드는 기술 프리뷰로만 사용할 수 있습니다.



참고

이 지원에는 Open vSwitch 2.9 이상이 필요합니다.

OVS 본드 모드	애플리케이션	참고	호환되는 LACP 옵션
active-backup	고가용성(active-passive)		활성, 수동 또는 꺼짐

balance-slb	처리량 증가 (활성-활성)	• 성능은 패킷당 추가 구문 분석의 영향을 받습니다. • vhost-user 잠금 경합이 발생할 가능성이 있습니다.	활성, 수동 또는 꺼짐
balance-tcp (기술 프리뷰만 해당)	권장되지 않음 (활성-활성)	• L4 해싱에 필요한 재시도는 성능에 영향을 미칩니다. • balance-slb와 마찬가지로 성능은 패킷당 추가 구문 분석의 영향을 받으며 vhost-user 잠금 경합이 발생할 가능성이 있습니다. • LACP를 활성화해야 합니다.	활성 또는 패시브

다음 예와 같이 BondInterfaceOvsOptions 매개변수를 사용하여 네트워크 환경 파일에서 본딩된 인터페이스를 구성할 수 있습니다.

```
parameter_defaults:
  BondInterfaceOvsOptions: "bond_mode=balance-slb"
```

13.3. LINUX 본딩 옵션

네트워크 인터페이스 템플릿에서 Linux 본딩과 함께 LACP를 사용할 수 있습니다.

```
    - type: linux_bond
      name: bond1
      members:
      - type: interface
        name: nic2
      - type: interface
        name: nic3
      bonding_options: "mode=802.3ad lacp_rate=[fast|slow] updelay=1000 miimon=100"
```

- 모드 - LACP 활성화.
- lacp_rate - LACP 패킷이 1초 또는 30초마다 전송되는지 여부를 정의합니다.
- updelay - 트래픽에 사용하기 전에 인터페이스를 활성화해야 하는 최소 시간을 정의합니다 (이로 인해 포트가 중단을 줄이는 데 도움이 됩니다).
- miimon - 드라이버의 MIIMON 기능을 사용하여 포트 상태를 모니터링하는 데 사용되는 간격 (밀리초)입니다.

Linux 본딩 옵션에 대한 자세한 내용은 7.7.1을 참조하십시오. *Red Hat Enterprise Linux 7 네트워킹 가이드* 의 본딩 모듈 지시문

13.4. OVS 본딩 옵션

다음 표에서는 이러한 옵션과 하드웨어에 따라 몇 가지 대안을 설명합니다.

표 13.1. 본딩 옵션

bond_mode=balance-slb	트래픽 패턴이 변경될 때 주기적으로 리밸런싱을 통해 소스 MAC 주소와 출력 VLAN을 기반으로 밸런싱을 조정합니다. **balance-slb** 와 결합하면 원격 스위치의 지식이나 협력 없이 제한된 형태의 부하 분산이 가능합니다. SLB는 각 소스 MAC 및 VLAN 쌍을 링크에 할당하고 해당 링크를 통해 해당 MAC 및 VLAN의 모든 패킷을 전송합니다. 이 모드에서는 소스 MAC 주소와 VLAN 번호를 기반으로 간단한 해싱 알고리즘을 사용하고 트래픽 패턴이 변경될 때 주기적으로 리밸런싱을 수행합니다. 이 모드는 Linux 본딩 드라이버에서 사용하는 모드 2 본딩과 유사합니다. 이 모드는 스위치가 LACP를 사용하도록 구성되지 않은 경우에도 부하 분산을 제공하는 데 사용할 수 있습니다.
bond_mode=active-backup	이 모드에서는 활성 연결이 실패하면 대기 NIC가 네트워크 작업을 다시 시작하는 활성/대기 페일오버를 제공합니다. 실제 스위치에는 하나의 MAC 주소만 표시됩니다. 이 모드에서는 특별한 스위치 지원이나 구성이 필요하지 않으며 링크가 별도의 스위치에 연결될 때 작동합니다. 이 모드에서는 로드 밸런싱을 제공하지 않습니다.

lacp=[active|passive|off]	LACP(링크 집계 제어 프로토콜) 동작을 제어합니다. 특정 스위치만 LACP를 지원합니다. 스위치가 LACP를 지원하지 않는 경우 **bond_mode=balance-slb** 또는 **bond_mode=active-backup** 을 사용합니다.
other-config:lacp-fallback-ab=true	대체로 bond_mode=active-backup으로 전환하도록 LACP 동작을 설정합니다.
other_config:lacp-time=[fast|slow]	LACP 하트비트를 1초(빠른) 또는 30초(slow)로 설정합니다. 기본값은 느립니다.
other_config:bond-detect-mode=[miimon|carrier]	miimon heartbeats(miimon) 또는 모니터 캐리어(carrier)를 사용하도록 링크 탐지를 설정합니다. 기본값은 캐리어입니다.
other_config:bond-miimon-interval=100	miimon을 사용하는 경우 하트비트 간격(밀리초)을 설정합니다.
bond_updelay=1000	플랩을 방지하려면 링크를 활성화해야 합니다.
other_config:bond-rebalance-interval=10000	본딩 멤버 간 리밸런싱 간격(밀리초)입니다. 본딩 멤버 간의 리밸런싱 흐름을 비활성화하려면 이 값을 0으로 설정합니다.

14장. 노드 배치 제어

director의 기본 동작은 일반적으로 프로필 태그를 기반으로 각 역할에 대해 노드를 임의로 선택하는 것입니다. 그러나 director는 특정 노드 배치를 정의하는 기능을 제공합니다. 이는 다음을 수행하는 유용한 방법입니다.

- controller-0, controller -1 등 특정 노드 ID 할당
- 사용자 정의 호스트 이름 할당
- 특정 IP 주소 할당
- 특정 가상 IP 주소 할당

참고

네트워크에 예측 가능한 IP 주소, 가상 IP 주소 및 포트를 수동으로 설정하면 풀 할당의 필요성이 줄어듭니다. 그러나 새 노드를 쉽게 확장할 수 있도록 각 네트워크에 대한 할당 풀을 유지하는 것이 좋습니다. 정적으로 정의된 모든 IP 주소가 할당 풀 외부에 속하는지 확인합니다. 할당 풀 설정에 대한 자세한 내용은 11.7절. “사용자 지정 네트워크 환경 파일”을 참조하십시오.

14.1. 특정 노드 ID 할당

이 절차에서는 특정 노드에 노드 ID를 할당합니다. 노드 ID의 예로는 controller-0,controller-1, compute -0,compute-1 등이 있습니다.

첫 번째 단계는 계산 스케줄러가 배포에 일치하는 노드별 기능으로 ID를 할당하는 것입니다. 예를 들면 다음과 같습니다.

```
openstack baremetal node set --property capabilities='node:controller-0,boot_option:local' <id>
```

그러면 기능 node:controller-0 이 노드에 할당됩니다. 모든 노드에 대해 0부터 시작하여 고유한 연속 인덱스를 사용하여 이 패턴을 반복합니다. 지정된 역할(Controller, Compute 또는 각 스토리지 역할)의 모든 노드에 동일한 방식으로 태그가 지정되었는지 확인합니다. 그러지 않으면 Compute 스케줄러가 기능과 올바르게 일치하지 않는지 확인합니다.

다음 단계는 스케줄러 힌트를 사용하여 각 노드의 기능과 일치시키는 **Heat** 환경 파일(예: **scheduler_hints_env.yaml**)을 생성하는 것입니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
```

이러한 스케줄러 힌트를 사용하려면 **Overcloud** 생성 중에 **overcloud deploy** 명령에 '**scheduler_hints_env.yaml**' 환경 파일을 포함합니다.

다음 매개 변수를 통해 각 역할에 대해 동일한 접근 방식을 사용할 수 있습니다.

- 컨트롤러 노드의 **ControllerSchedulerHints** 입니다.
- 컴퓨팅 노드의 **ComputeSchedulerHints** 입니다.
- 블록 스토리지 노드의 **BlockStorageSchedulerHints** 입니다.
- 오브젝트 스토리지 노드의 **ObjectStorageSchedulerHints** 입니다.
- **Ceph Storage** 노드의 **CephStorageSchedulerHints.**
- **[ROLE]** 사용자 정의 역할을 위한 스케줄 힌트입니다. **[ROLE]** 을 역할 이름으로 바꿉니다.

참고

노드 배치는 프로필 일치보다 우선합니다. 스케줄링 실패를 방지하려면 프로필 일치(컴퓨팅, 제어 등)용으로 설계된 플레이버가 아닌 배포에 기본 **baremetal** 플레이버를 사용합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy ... --control-flavor baremetal --compute-flavor baremetal
...
```

14.2. 사용자 지정 호스트 이름 할당

14.1절. “특정 노드 ID 할당” 의 노드 ID 설정과 함께 director는 각 노드에 특정 사용자 지정 호스트 이름을 할당할 수도 있습니다. 이 기능은 시스템이 있는 위치(예: rack2-row12)를 정의해야 하거나 인벤토리식별자 또는 사용자 지정 호스트 이름이 필요한 기타 상황을 정의해야 하는 경우에 유용합니다.

중요

노드 이름을 배포한 후에는 이름을 바꾸지 마십시오. 배포 후 노드 이름을 변경하면 인스턴스 관리 문제가 발생합니다.

노드 호스트 이름을 사용자 지정하려면 14.1절. “특정 노드 ID 할당” 의 ' scheduler_hints_env.yaml' 파일과 같은 환경 파일에서 HostnameMap 매개변수를 사용합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-compute-0: overcloud-compute-prod-abc-0
```

parameter_defaults 섹션에 HostnameMap 을 정의하고 각 매핑을 HostnameFormat 매개 변수(예: overcloud-controller-0)를 사용하여 정의하는 원래 호스트 이름으로 설정하고 두 번째값은 해당 노드에 대해 원하는 사용자 지정 호스트 이름(예: overcloud-controller-prod-123-0)입니다.

노드 ID 배치와 함께 이 방법을 사용하면 각 노드에 사용자 지정 호스트 이름이 있습니다.

14.3. 예측 가능한 IP 할당

결과 환경을 추가로 제어하기 위해 director는 각 네트워크에서 특정 IP를 사용하여 Overcloud 노드를 할당할 수 있습니다. 코어 Heat 템플릿 컬렉션에서 environments/ips-from-pool-all.yaml 환경 파일을 사용합니다.

이 파일을 stack 사용자의 templates 디렉터리에 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/ips-from-pool-all.yaml ~/templates/.
```

ips-from-pool-all.yaml 파일에는 두 개의 주요 섹션이 있습니다.

첫 번째는 기본값을 재정의하는 **resource_registry** 참조 집합입니다. 이를 통해 **director**에 노드 유형의 지정된 포트에 특정 **IP**를 사용하도록 지시합니다. 각 리소스를 수정하여 해당 템플릿의 절대 경로를 사용합니다. 예를 들면 다음과 같습니다.

```
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external_from_pool.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api_from_pool.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/storage_from_pool.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt_from_pool.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant_from_pool.yaml
```

기본 구성은 사전 할당된 **IP**를 사용하도록 모든 노드 유형의 모든 네트워크를 설정합니다. 특정 네트워크 또는 노드 유형이 기본 **IP** 할당을 대신 사용하도록 허용하려면 환경 파일에서 해당 노드 유형 또는 네트워크와 관련된 **resource_registry** 항목을 제거하면 됩니다.

두 번째 섹션은 실제 **IP** 주소가 할당되는 **parameter_defaults**입니다. 각 노드 유형에는 연결된 매개변수가 있습니다.

- 컨트롤러 노드의 **ControllerIP.**
- 컴퓨팅 노드 의 **ComputeIP.**
- **Ceph Storage** 노드의 **CephStorageIP.**
- 블록 스토리지 노드 의 **BlockStorageIP.**
- 오브젝트 스토리지 노드의 **SwiftStorageIPs.**
- **[ROLE]**사용자 정의 역할을 위한 **IP**입니다. **[ROLE]** 을 역할 이름으로 바꿉니다.

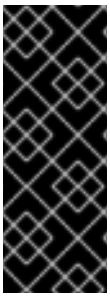
각 매개 변수는 네트워크 이름에서 주소 목록에 대한 맵입니다. 각 네트워크 유형에는 해당 네트워크에 노드가 있을 만큼의 주소가 있어야 합니다. **director**에서 주소를 순서대로 할당합니다. 각 유형의 첫 번째 노드는 각 목록에서 첫 번째 주소를 수신하고 두 번째 노드는 각 목록에서 두 번째 주소를 수신합니다.

예를 들어 오버클라우드에 세 개의 **Ceph Storage** 노드가 포함된 경우 **CephStorageIPs** 매개변수는 다음과 같을 수 있습니다.

```
CephStorageIPs:
  storage:
  - 172.16.1.100
  - 172.16.1.101
  - 172.16.1.102
  storage_mgmt:
  - 172.16.3.100
  - 172.16.3.101
  - 172.16.3.102
```

첫 번째 **Ceph Storage** 노드는 두 개의 주소를 수신합니다. **172.16.1.100** 및 **172.16.3.100**. 두 번째는 **172.16.1.101** 및 **172.16.3.101**을 수신하고 세 번째는 **172.16.1.102** 및 **172.16.3.102**를 수신합니다. 동일한 패턴은 다른 노드 유형에 적용됩니다.

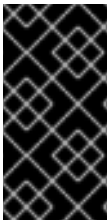
선택한 **IP** 주소가 네트워크 환경 파일에 정의된 각 네트워크의 할당 풀 외부에 속하는지 확인합니다(11.7절. “사용자 지정 네트워크 환경 파일”참조). 예를 들어 **internal_api** 할당이 **InternalApiAllocationPools** 범위 밖에 속하는지 확인합니다. 이렇게 하면 자동으로 선택한 **IP**와의 충돌이 방지됩니다. 마찬가지로 표준 예측 가능한 **VIP** 배치(14.4절. “예측 가능한 가상 IP 할당”참조) 또는 외부 로드 밸런싱에 대한 **IP** 할당이 **VIP** 구성과 충돌하지 않는지 확인합니다(23.2절. “외부 로드 밸런싱 구성”참조).



중요

Overcloud 노드가 삭제되면 **IP** 목록의 항목을 제거하지 마십시오. **IP** 목록은 기본 **Heat** 인덱스를 기반으로 하며, 노드를 삭제해도 변경되지 않습니다. 목록에 지정된 항목이 더 이상 사용되지 않음을 나타내려면 **IP** 값을 **DELETED** 또는 **UNUSED** 와 같은 값으로 바꿉니다. 항목은 변경 또는 추가만 **IP** 목록에서 제거해서는 안 됩니다.

배포 중에 이 구성을 적용하려면 **openstack overcloud deploy** 명령을 사용하여 **ips-from-pool-all.yaml** 환경 파일을 포함합니다.



중요

네트워크 분리를 사용하는 경우 **network -isolation.yaml** 파일 뒤에 **ips-from-pool-all.yaml** 파일을 포함합니다.

예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/ips-from-pool-all.yaml \
  [OTHER OPTIONS]
```

14.4. 예측 가능한 가상 IP 할당

director는 각 노드에 대해 예측 가능한 **IP** 주소를 정의하는 것 외에도 클러스터형 서비스에 대해 예측 가능한 **VIP(**가상 **IP)**를 정의하는 유사한 기능도 제공합니다. 이 작업을 수행하려면 **11.7**절**. "**사용자 지정 네트워크 환경 파일**"** 에서 네트워크 환경 파일을 편집하고 **parameter_defaults** 섹션에 **VIP** 매개변수를 추가합니다.

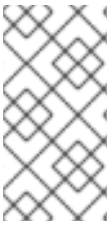
```
parameter_defaults:
  ...
  # Predictable VIPs
  ControlFixedIPs: [{'ip_address':'192.168.201.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.0.9'}]
  PublicVirtualFixedIPs: [{'ip_address':'10.1.1.9'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.18.0.9'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.19.0.9'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.16.0.8'}]
```

해당 할당 풀 범위 외부에서 이러한 **IP**를 선택합니다. 예를 들어 **InternalApi AllocationPools** 범위에 없는 **InternalApiVirtualFixedIP s**의 **IP** 주소를 선택합니다.

이 단계는 기본 내부 로드 밸런싱 구성을 사용하는 오버클라우드 전용입니다. 외부 로드 밸런서에 **VIP**를 할당하는 경우 **Overcloud**용 전용 **External Load Balancing** 가이드의 절차를 사용합니다.

15장. 오버클라우드 공개 엔드 포인트에서 SSL/TLS 활성화

기본적으로 오버클라우드는 암호화되지 않은 엔드포인트를 서비스에 사용합니다. 즉, 공용 API 엔드포인트에 SSL/TLS를 활성화하려면 오버클라우드 구성에 추가 환경 파일이 필요합니다. 다음 장에서는 SSL/TLS 인증서를 구성하고 오버클라우드 생성의 일부로 포함하는 방법을 보여줍니다.



참고

이 프로세스는 공용 API 엔드포인트에 대해 SSL/TLS만 활성화합니다. Internal 및 Admin API는 암호화되지 않은 상태로 유지됩니다.

이 프로세스를 수행하려면 공용 API의 엔드포인트를 정의하기 위해 네트워크 격리가 필요합니다.

15.1. 서명 호스트 초기화

서명 호스트는 새 인증서를 생성하고 인증 기관을 통해 서명하는 호스트입니다. 선택한 서명 호스트에서 SSL 인증서를 생성한 적이 없는 경우 새 인증서에 서명할 수 있도록 호스트를 초기화해야 할 수 있습니다.

/etc/pki/CA/index.txt 파일은 서명된 모든 인증서의 레코드를 저장합니다. 이 파일이 있는지 확인합니다. 없는 경우 빈 파일을 생성합니다.

```
$ sudo touch /etc/pki/CA/index.txt
```

/etc/pki/CA/serial 파일은 서명할 다음 인증서에 사용할 다음 일련번호를 식별합니다. 이 파일이 있는지 확인합니다. 존재하지 않는 경우 새 시작 값으로 새 파일을 생성합니다.

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

15.2. 인증 기관 생성

일반적으로는 외부 인증 기관을 통해 SSL/TLS 인증서에 서명합니다. 경우에 따라 고유한 인증 기관을 사용하려고 할 수 있습니다. 예를 들어 내부 전용 인증 기관을 사용할 수 있습니다.

예를 들어 인증 기관 역할을 할 키와 인증서 쌍을 생성합니다.

```
$ sudo openssl genrsa -out ca.key.pem 4096
$ sudo openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

openssl req 명령은 기관에 대한 특정 세부 정보를 요청합니다. 다음 세부 정보를 입력합니다.

이렇게 하면 **ca.crt.pem** 이라는 인증 기관 파일이 생성됩니다.

15.3. 클라이언트에 인증 기관 추가

외부 클라이언트가 **SSL/TLS**를 사용하여 통신하려는 경우 **Red Hat OpenStack Platform** 환경에 액세스해야 하는 각 클라이언트에 인증 기관 파일을 복사합니다. 클라이언트에 복사되고 나면 클라이언트에서 다음 명령을 실행하여 인증 기관 신뢰 번들에 추가합니다.

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

예를 들어 언더클라우드에는 생성 중에 오버클라우드 엔드포인트와 통신할 수 있도록 인증 기관 파일 사본이 필요합니다.

15.4. SSL/TLS 키 생성

다음 명령을 실행하여 언더클라우드 또는 오버클라우드 인증서를 생성하는 데 다른 지점에서 사용하는 **SSL/TLS** 키**(server.key.pem)**를 생성합니다.

```
$ openssl genrsa -out server.key.pem 2048
```

15.5. SSL/TLS 인증서 서명 요청 생성

다음 절차에서는 오버클라우드에 대한 인증서 서명 요청을 생성합니다. 사용자 지정을 위해 기본 **OpenSSL** 구성 파일을 복사합니다.

```
$ cp /etc/pki/tls/openssl.cnf .
```

사용자 지정 **openssl.cnf** 파일을 편집하고 오버클라우드에 사용할 **SSL** 매개변수를 설정합니다. 수정할 매개변수 유형의 예제는 다음과 같습니다.

```
[req]
distinguished_name = req_distinguished_name
```

```
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 10.0.0.1
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 = 10.0.0.1
DNS.1 = 10.0.0.1
DNS.2 = myovercloud.example.com
```

commonName_default 를 다음 중 하나로 설정합니다.

- **IP**를 사용하여 **SSL/TLS**를 통해 액세스하는 경우 공용 **API**에 대해 가상 **IP**를 사용합니다. 환경 파일에서 **PublicVirtualFixedIPs** 매개변수를 사용하여 이 **VIP**를 설정합니다. 자세한 내용은 **14.4**절. “예측 가능한 가상 **IP** 할당”의 내용을 참조하십시오. 예측 가능한 **VIP**를 사용하지 않는 경우 **director**는 **ExternalAllocationPools** 매개변수에 정의된 범위에서 첫 번째 **IP** 주소를 할당합니다.

- 정규화된 도메인 이름을 사용하여 **SSL/TLS**를 통해 액세스하는 경우 대신 도메인 이름을 사용합니다.

동일한 공용 **API IP** 주소를 **IP** 항목으로 포함하고 **alt_names** 섹션에 **DNS** 항목을 포함합니다. **DNS**를 사용하는 경우 서버의 호스트 이름을 동일한 섹션에 **DNS** 항목으로 포함합니다. **openssl.cnf** 에 대한 자세한 내용을 보려면 **man openssl.cnf** 를 실행합니다.

다음 명령을 실행하여 인증서 서명 요청**(server.csr.pem)**을 생성합니다.

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

-key 옵션에 대해 **15.4절. "SSL/TLS 키 생성"** 에서 생성한 **SSL/TLS** 키를 포함해야 합니다.

다음 섹션에서 **server.csr.pem** 파일을 사용하여 **SSL/TLS** 인증서를 생성합니다.

15.6. 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 명령에서는 다음을 사용합니다.

- **v3** 확장을 지정하는 구성 파일입니다. 이 파일을 **-config** 옵션으로 추가합니다.
- 인증서를 생성하고 인증 기관을 통해 서명하기 위한 **15.5절. "SSL/TLS 인증서 서명 요청 생성"** 의 인증서 서명 요청입니다. 이 파일을 **-in** 옵션으로 추가합니다.
- **15.2절. "인증 기관 생성"** 에서 생성한 인증 기관. 이 인증 기관을 **-cert** 옵션으로 추가합니다.
- **15.2절. "인증 기관 생성"** 에서 생성한 인증 기관 개인 키입니다. 이 파일을 **-keyfile** 옵션으로 추가합니다.

그러면 **server.crt.pem**이라는 인증서가 생성됩니다. 이 인증서를 **15.4절. "SSL/TLS 키 생성"** 의 **SSL/TLS** 키와 함께 사용하여 **SSL/TLS**를 활성화합니다.

15.7. SSL/TLS 활성화

Heat 템플릿 컬렉션에서 **enable-tls.yaml** 환경 파일을 복사합니다.

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml ~/templates/.
```

이 파일을 편집하여 해당 매개변수에 대해 다음과 같이 변경합니다.

SSLCertificate

인증서 파일(**server.crt.pem**)의 내용을 **SSLCertificate** 매개 변수에 복사합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

SSLIntermediateCertificate

중간 인증서가 있는 경우 중간 인증서의 내용을 **SSLIntermediateCertificate** 매개변수로 복사합니다.

```
parameter_defaults:
  SSLIntermediateCertificate: |
    -----BEGIN CERTIFICATE-----
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvpBCwUAMFgxCzAJB
    ...
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQE
    -----END CERTIFICATE-----
```

중요

인증서 콘텐츠에는 모든 새 행에 대해 동일한 들여쓰기 수준이 필요합니다.

SSLKey

개인 키(**server.key.pem**)의 콘텐츠를 **SSLKey** 매개 변수에 복사합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO9hkJZnGP6qb6wtYUoy1bVP7
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4XpZUC7yhqPaU
    -----END RSA PRIVATE KEY-----
```

중요

개인 키 콘텐츠에는 모든 새 행에 대해 동일한 들여쓰기 수준이 필요합니다.

OS::TripleO::NodeTLSData

OS::TripleO::NodeTLSData: 의 리소스 경로를 절대 경로로 변경합니다.

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

15.8. 루트 인증서 삽입

인증서 서명자가 오버클라우드 이미지의 기본 신뢰 저장소에 없는 경우 인증 기관을 오버클라우드 이미지에 삽입해야 합니다. **heat** 템플릿 컬렉션에서 **inject-trust-anchor.yaml** 환경 파일을 복사합니다.

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

이 파일을 편집하여 해당 매개변수에 대해 다음과 같이 변경합니다.

SSLRootCertificate

루트 인증 기관 파일(**ca.crt.pem**)의 콘텐츠를 **SSLRootCertificate** 매개 변수에 복사합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

중요

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

OS::TripleO::NodeTLSCAData

OS::TripleO::NodeTLSCAData: 의 리소스 경로를 절대 경로로 변경합니다.

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

여러 **CA**를 삽입하려면 **inject-trust-anchor-hiera.yaml** 환경 파일을 사용할 수 있습니다. 예를 들어 언더클라우드와 오버클라우드 모두에 대한 **CA**를 동시에 삽입할 수 있습니다.

```
parameter_defaults:
  CAMap:
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        ... cert content ...
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        ... cert content ...
        -----END CERTIFICATE-----
```

15.9. DNS 끝점 구성

DNS 호스트 이름을 사용하여 **SSL/TLS**를 통해 오버클라우드에 액세스하는 경우 **custom-domain.yaml** 파일을 **/home/stack/templates** 에 복사해야 합니다. 이 파일은 **/usr/share/tripleo-heat-templates/environments/predictable-placement/** 에서 찾을 수 있습니다.

1. 필요한 경우 사용자 지정 네트워크의 매개 변수를 추가하여 모든 필드에 대해 호스트 및 도메인 이름을 구성합니다.

참고

이 환경 파일이 초기 배포에 포함되지 않은 경우 **TLS-everywhere** 아키텍처로 재배포할 수 없습니다.

```
# title: Custom Domain Name
# description: |
#   This environment contains the parameters that need to be set in order to
#   use a custom domain name and have all of the various FQDNs reflect it.
parameter_defaults:
  # The DNS domain used for the hosts. This must match the overcloud_domain_name
```

```
configured on the undercloud.
  # Type: string
  CloudDomain: localdomain

  # The DNS name of this cloud. E.g. ci-overcloud.tripleo.org
  # Type: string
  CloudName: overcloud.localdomain

  # The DNS name of this cloud's provisioning network endpoint. E.g. 'ci-
overcloud.ctlplane.tripleo.org'.
  # Type: string
  CloudNameCtlplane: overcloud.ctlplane.localdomain

  # The DNS name of this cloud's internal_api endpoint. E.g. 'ci-
overcloud.internalapi.tripleo.org'.
  # Type: string
  CloudNameInternal: overcloud.internalapi.localdomain

  # The DNS name of this cloud's storage endpoint. E.g. 'ci-overcloud.storage.tripleo.org'.
  # Type: string
  CloudNameStorage: overcloud.storage.localdomain

  # The DNS name of this cloud's storage_mgmt endpoint. E.g. 'ci-
overcloud.storagemgmt.tripleo.org'.
  # Type: string
  CloudNameStorageManagement: overcloud.storagemgmt.localdomain
```

2. **새 환경 파일이나 기존 환경 파일에서 매개 변수 기본값 아래 사용할 DNS 서버 목록을 추가합니다.**

```
parameter_defaults:
  DnsServers: ["10.0.0.254"]
  ....
```

15.10. 오버클라우드 생성 중 환경 파일 추가

배포 명령(openstack overcloud deploy)은 -e 옵션을 사용하여 환경 파일을 추가합니다. 이 섹션의 환경 파일을 다음 순서로 추가합니다.

- **SSL/TLS(enable-tls.yaml)를 활성화하는환경 파일**
- **DNS 호스트 이름을 설정할 환경 파일 (cloudname.yaml)**
- **루트 인증 기관을 삽입할 환경 파일(inject-trust-anchor.yaml)**

- 공용 끝점 매핑을 설정하는 환경 파일은 다음과 같습니다.
  - 공용 끝점 액세스에 **DNS** 이름을 사용하는 경우 **/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml**을 사용합니다.
  - 공용 엔드포인트 액세스에 **IP** 주소를 사용하는 경우 **/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-ip.yaml**을 사용합니다.

예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates [...] -e /home/stack/templates/enable-tls.yaml -e ~/templates/cloudname.yaml -e ~/templates/inject-trust-anchor.yaml -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

15.11. SSL/TLS 인증서 업데이트

나중에 인증서를 업데이트해야 하는 경우:

- **enable-tls.yaml** 파일을 편집하고 **SSLCertificate, SSL Key** 및 **SSLIntermediateCertificate** 매개변수를 업데이트합니다.
- 인증 기관이 변경된 경우 **inject-trust-anchor.yaml** 파일을 편집하고 **SSLRootCertificate** 매개변수를 업데이트합니다.

새 인증서 내용이 준비되면 배포 명령을 다시 실행합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates [...] -e /home/stack/templates/enable-tls.yaml -e ~/templates/cloudname.yaml -e ~/templates/inject-trust-anchor.yaml -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

16장. ID 관리를 사용하여 내부 및 공개 끝점에서 SSL/TLS 활성화

특정 오버클라우드 끝점에서 **SSL/TLS**를 활성화할 수 있습니다. 필요한 인증서 수로 인해 **director**는 **Red Hat IdM(Identity Management)** 서버와 통합되어 인증 기관 역할을 하며 오버클라우드 인증서를 관리합니다. 이 프로세스에는 **novajoin** 을 사용하여 **Overcloud** 노드를 **IdM** 서버에 등록해야 합니다.

OpenStack 구성 요소에서 **TLS** 지원 상태를 확인하려면 **TLS 활성화 상태 매트릭스** 를 참조하십시오.

16.1. 언더클라우드를 CA에 추가합니다

오버클라우드를 배포하기 전에 **CA(인증 기관)**에 언더클라우드를 추가해야 합니다.

1. 언더클라우드 노드에서 **python-novajoin** 패키지를 설치합니다.

```
$ sudo yum install python-novajoin
```

2. 언더클라우드 노드에서 **novajoin-ipa-setup** 스크립트를 실행하여 배포에 맞게 값을 조정합니다.

```
$ sudo /usr/libexec/novajoin-ipa-setup \
  --principal admin \
  --password <IdM admin password> \
  --server <IdM server hostname> \
  --realm <overcloud cloud domain (in upper case)> \
  --domain <overcloud cloud domain> \
  --hostname <undercloud hostname> \
  --precreate
```

다음 섹션에서는 결과로 생성된 **OTP(One-Time Password)**를 사용하여 언더클라우드를 등록합니다.

16.2. IDM에 언더클라우드 추가

이 절차에서는 언더클라우드를 **IdM**에 등록하고 **novajoin**을 구성합니다. **undercloud.conf ([DEFAULT]** 섹션 내)에서 다음 설정을 구성합니다.

1. **novajoin** 서비스는 기본적으로 비활성화되어 있습니다. 활성화하려면 다음을 수행합니다.

```
[DEFAULT]
enable_novajoin = true
```

2. **IdM**을 사용하여 언더클라우드 노드를 등록하려면 **OTP(**일회 암호**)**를 설정해야 합니다**.**

```
ipa_otp = <otp>
```

3. **neutron**의 **DHCP** 서버에서 제공하는 오버클라우드의 도메인 이름이 **IdM** 도메인**(**소문자의 **kerberos** 영역**)**과 일치하는지 확인합니다**.**

```
overcloud_domain_name = <domain>
```

4. 언더클라우드에 적절한 호스트 이름을 설정합니다**.**

```
undercloud_hostname = <undercloud FQDN>
```

5. **IdM**을 언더클라우드의 이름 서버로 설정합니다**.**

```
undercloud_nameservers = <IdM IP>
```

6. 대규모 환경의 경우 **novajoin** 연결 시간 제한 값을 검토해야 합니다**. undercloud.conf**에서 **undercloud -timeout.yaml**이라는 새 파일에 참조를 추가합니다**.**

```
hieradata_override = /home/stack/undercloud-timeout.yaml
```

undercloud-timeout.yaml 에 다음 옵션을 추가합니다**.** 시간 제한 값을 초 단위로 지정할 수 있습니다**(**예**: 5):**

```
nova::api::vendordata_dynamic_connect_timeout: <timeout value>
nova::api::vendordata_dynamic_read_timeout: <timeout value>
```

7. **undercloud.conf** 파일을 저장합니다**.**

8. 언더클라우드 배포 명령을 실행하여 기존 언더클라우드에 변경 사항을 적용합니다**.**

```
$ openstack undercloud install
```

검증

1. 키 탭 파일에서 언더클라우드의 키 항목을 확인합니다.

```
[root@undercloud-0 ~]# klist -kt
Keytab name: FILE:/etc/krb5.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   1 04/28/2020 12:22:06 host/undercloud-0.redhat.local@REDHAT.LOCAL
   1 04/28/2020 12:22:06 host/undercloud-0.redhat.local@REDHAT.LOCAL


[root@undercloud-0 ~]# klist -kt /etc/novajoin/krb5.keytab
Keytab name: FILE:/etc/novajoin/krb5.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   1 04/28/2020 12:22:26 nova/undercloud-0.redhat.local@REDHAT.LOCAL
   1 04/28/2020 12:22:26 nova/undercloud-0.redhat.local@REDHAT.LOCAL
```

2. 호스트 원칙을 사용하여 시스템 **/etc/krb.keytab** 파일을 테스트합니다.

```
[root@undercloud-0 ~]# kinit -k
[root@undercloud-0 ~]# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: host/undercloud-0.redhat.local@REDHAT.LOCAL

Valid starting       Expires              Service principal
05/04/2020 10:34:30  05/05/2020 10:34:30  krbtgt/REDHAT.LOCAL@REDHAT.LOCAL

[root@undercloud-0 ~]# kdestroy
Other credential caches present, use -A to destroy all
```

3. **nova** 원칙을 사용하여 **novajoin /etc/novajoin/krb.keytab** 파일을 테스트합니다.

```
[root@undercloud-0 ~]# kinit -kt /etc/novajoin/krb5.keytab 'nova/undercloud-
0.redhat.local@REDHAT.LOCAL'
[root@undercloud-0 ~]# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: nova/undercloud-0.redhat.local@REDHAT.LOCAL

Valid starting       Expires              Service principal
05/04/2020 10:39:14  05/05/2020 10:39:14  krbtgt/REDHAT.LOCAL@REDHAT.LOCAL
```

16.3. 오버클라우드 DNS 설정

IdM 환경을 자동으로 감지하고 등록하기 쉬운 경우 **DNS** 서버로 **IdM**을 사용하는 것이 좋습니다.

1. 언더클라우드에 연결합니다.

```
$ source ~/stackrc
```

2. IdM을 DNS 이름 서버로 사용하도록 컨트롤 플레인 서브넷을 구성합니다.

```
$ openstack subnet set ctlplane-subnet --dns-nameserver  <idm_server_address>
```

3. IdM 서버를 사용하도록 환경 파일에서 DnsServers 매개변수를 설정합니다.

```
parameter_defaults:
  DnsServers: ["<idm_server_address>"]
```

일반적으로 이 매개변수는 사용자 지정 network-environment.yaml 파일에 정의됩니다.

16.4. NOVAJOIN을 사용하도록 오버클라우드 설정

1. IdM 통합을 활성화하려면 /usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml 환경 파일의 사본을 생성합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/predictable-
placement/custom-domain.yaml \
  /home/stack/templates/custom-domain.yaml
```

2. /home/stack/templates/custom-domain.yaml 환경 파일을 편집하고 배포에 맞게 CloudDomain 및 CloudName* 값을 설정합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  CloudDomain: lab.local
  CloudName: overcloud.lab.local
  CloudNameInternal: overcloud.internalapi.lab.local
  CloudNameStorage: overcloud.storage.lab.local
  CloudNameStorageManagement: overcloud.storagemgmt.lab.local
  CloudNameCtlplane: overcloud.ctlplane.lab.local
```

3. 오버클라우드 배포 프로세스에 다음 환경 파일을 포함합니다.

- /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-

tls.yaml

- **/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml**

- **/home/stack/templates/custom-domain.yaml**

예를 들면 다음과 같습니다.

```
openstack overcloud deploy \
  --templates \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
   -e /home/stack/templates/custom-domain.yaml \
```

결과적으로 배포된 Overcloud 노드가 IdM을 사용하여 자동으로 등록됩니다.

4. **이렇게 하면 내부 엔드포인트에 TLS만 설정됩니다. 외부 엔드포인트의 경우 /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml 환경 파일과 함께 TLS를 추가하는 일반 수단을 사용할 수 있습니다(사용자 정의 인증서 및 키를 추가하려면 수정해야 함). 결과적으로 openstack deploy 명령은 다음과 유사합니다.**

```
openstack overcloud deploy \
  --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e /home/stack/templates/custom-domain.yaml \
  -e /home/stack/templates/enable-tls.yaml
```

5. **또는 IdM을 사용하여 공용 인증서를 발급할 수도 있습니다. 이 경우 /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml 환경 파일을 사용해야 합니다. 예를 들면 다음과 같습니다.**

```
openstack overcloud deploy \
  --templates \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
```

```
  -e /home/stack/templates/custom-domain.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-
certmonger.yaml
```

17장. TLS를 사용하도록 기존 배포 변환

TLS 암호화를 사용하도록 기존 오버클라우드 및 언더클라우드 끝점을 구성할 수 있습니다. 이 접근 방식에서는 novajoin 을 사용하여 배포를 Red Hat IdM(Identity Management)과 통합하고 DNS, Kerberos 및 certmonger에 액세스할 수 있습니다. 각 오버클라우드 노드는 certmonger 클라이언트를 사용하여 각 서비스의 인증서를 검색합니다.

TLS에 대한 자세한 내용은 보안 및 강화 가이드를 참조하십시오.

17.1. 요구 사항

- Red Hat OpenStack Platform 13의 경우 버전 z8 이상을 실행해야 합니다.
- 기존 IdM 배포가 있어야 하며 OpenStack 배포에 DNS 서비스를 제공해야 합니다.
- 기존 배포에서는 공용 엔드포인트에 FQDN을 사용해야 합니다. 기본 구성은 IP 주소 기반 엔드포인트를 사용할 수 있으므로 IP 주소 기반 인증서를 생성합니다. 이러한 단계를 진행하기 전에 FQDN으로 변경해야 합니다.

중요

이 절차 기간에는 오버클라우드 및 Undercloud 서비스를 사용할 수 없습니다.

17.2. 끝점 검토

기본적으로 기존 Red Hat OpenStack Platform 13 오버클라우드는 TLS를 사용하여 특정 끝점을 암호화하지 않습니다. 예를 들어 이 출력에는 https 대신 http 를 사용하는 URL이 포함되어 있습니다. 이러한 URL은 암호화되지 않습니다.

```
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------+
| ID                               | Region    | Service Name | Service Type | Enabled | Interface | URL                              |
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------+
| 0ad11e943e1f4ff988650cfba57b4031 | regionOne | nova         | compute      | True    | internal  | http://172.16.2.17:8774/v2.1     |
| 1413eb9ef38a45b8bee1bee1b0dfe744 | regionOne | swift        | object-store | True    | public    | https://overcloud.lab.local:13808/v1/AUTH_%(tenant_id)s |
| 1a54f13f212044b0a20468861cd06f85 | regionOne | neutron      | network      | True    | public    |
```

```
https://overcloud.lab.local:13696                    |
| 3477a3a052d2445697bb6642a8c26a91 | regionOne | placement    | placement    | True    | internal
| http://172.16.2.17:8778/placement                    |
| 3f56445c0dd14721ac830d6afb2c2cd4 | regionOne | nova         | compute      | True    | admin    |
http://172.16.2.17:8774/v2.1                         |
| 425b1773a55c4245bcbe3d051772ebba | regionOne | glance       | image        | True    | internal |
http://172.16.2.17:9292                              |
| 57cf09fa33ed446f8736d4228bdfa881 | regionOne | placement    | placement    | True    | public   |
https://overcloud.lab.local:13778/placement          |
| 58600f3751e54f7e9d0a50ba618e4c54 | regionOne | glance       | image        | True    | public   |
https://overcloud.lab.local:13292                    |
| 5c52f273c3284b068f2dc885c77174ca | regionOne | neutron      | network      | True    | internal |
http://172.16.2.17:9696                              |
| 8792a4dd8bbb456d9dea4643e57c43dc | regionOne | nova         | compute      | True    | public   |
https://overcloud.lab.local:13774/v2.1               |
| 94bbea97580a4c4b844478aad5a85e84 | regionOne | keystone     | identity     | True    | public   |
https://overcloud.lab.local:13000                    |
| acbf11b5c76d44198af49e3b78ffedcd | regionOne | swift        | object-store | True    | internal |
http://172.16.1.9:8080/v1/AUTH_%(tenant_id)s         |
| d4a1344f02a74f7ab0a50c5a7c13ca5c | regionOne | keystone     | identity     | True    | internal |
http://172.16.2.17:5000                              |
| d86c241dc97642419ddc12533447d73d | regionOne | placement    | placement    | True    | admin
| http://172.16.2.17:8778/placement                    |
| de7d6c34533e4298a2752852427a7030 | regionOne | glance       | image        | True    | admin    |
http://172.16.2.17:9292                              |
| e82086062ebd4d4b9e03c7f1544bdd3b | regionOne | swift        | object-store | True    | admin    |
http://172.16.1.9:8080                               |
| f8134cd9746247bca6a06389b563c743 | regionOne | keystone     | identity     | True    | admin    |
http://192.168.24.6:35357                            |
| fe29177bd29545ca8fdc0c777a7cf03f | regionOne | neutron      | network      | True    | admin    |
http://172.16.2.17:9696                              |
+----------------------------------+-----------+--------------+--------------+---------+-----------+------------------
-----------------------------------+
```

다음 섹션에서는 **TLS**를 사용하여 이러한 엔드포인트를 암호화하는 방법에 대해 설명합니다.

17.3. 알려진 문제에 대한 해결 방법 적용

현재 **TLS Everywhere** 내부 업그레이드에 대한 알려진 문제가 있으며 이로 인해 오버클라우드 노드는 **IdM**에 등록할 수 없습니다. 이 문제를 해결하려면 오버클라우드 배포를 실행하기 전에 모든 오버클라우드 노드에서 **/etc/ipa/ca.crt/** 를 삭제합니다. 자세한 내용은 **https://bugzilla.redhat.com/show_bug.cgi?id=1732564** 의 내용을 참조하십시오.

예를 들어 다음 스크립트는 해결 방법을 적용하는 한 가지 방법입니다. 배포에 맞게 수정해야 할 수도 있습니다.

```
[stack@undercloud-0 ~]$ vi rm-ca.crt-dir.sh
#!/bin/bash
```

```
source /home/stack/stackrc
NODES=$(openstack server list -f value -c Networks|sed s/ctlplane=//g)

for NODE in $NODES
do
   ssh heat-admin@$NODE sudo rm -rf /etc/ipa/ca.crt/
Done

[stack@undercloud-0 ~]$ bash rm-ca.crt-dir.sh
```

17.4. TLS를 사용하도록 끝점 구성

이 섹션에서는 기존 배포에 대해 **TLS** 엔드포인트 암호화를 활성화하는 방법과 엔드포인트가 올바르게 구성되었는지 확인하는 방법을 설명합니다.

어디에서든 **TLS**를 활성화할 때 도메인의 구조에 따라 다양한 업그레이드 경로를 사용할 수 있습니다. 다음 예제에서는 샘플 도메인 이름을 사용하여 업그레이드 경로를 설명합니다.

- 기존 공용 엔드포인트 인증서를 재사용하고, **Overcloud** 도메인(**lab.local**)이 **IdM** 도메인(**lab.local**)과 일치하는 내부 및 관리 엔드포인트의 모든 위치에서 **TLS**를 활성화합니다.
- **IdM**에서 새 공용 엔드포인트 인증서를 발급하고, 오버클라우드 도메인(**lab.local**)이 **IdM** 도메인(**lab.local**)과 일치하는 내부 및 관리 끝점의 모든 위치에서 **TLS**를 사용하도록 허용합니다.
- 기존 공용 엔드포인트 인증서를 재사용하고, **Overcloud** 도메인(**site1.lab.local**)이 **IdM** 도메인(**lab.local**)의 하위 도메인인 내부 및 관리 끝점의 모든 위치에서**TLS**를 활성화합니다.
- **IdM**에서 새 공용 엔드포인트 인증서를 발급하고, **Overcloud** 도메인(**site1.lab.local**)이 **IdM** 도메인(**lab.local**)의 하위 도메인인 내부 및 관리 끝점의 모든 위치에서**TLS**를 사용하도록 허용합니다.

이 섹션의 다음 절차에서는 위에서 설명한 다양한 조합을 사용하여 이 통합을 구성하는 방법에 대해 설명합니다.

17.4.1. IdM과 동일한 도메인을 사용하여 배포에 언더클라우드 통합 구성

다음 절차에서는 **IdM**과 동일한 도메인을 사용하는 배포에 언더클라우드 통합을 구성하는 방법을 설명합니다.

Red Hat OpenStack Platform은 novajoin 을 사용하여 Red Hat IdM(Identity Management)과 통합되어 암호화 인증서를 발급하고 관리합니다. 이 절차에서는 언더클라우드를 IdM에 등록하고, 토큰을 생성하고, 언더클라우드 설정에서 토큰을 활성화한 다음, 언더클라우드 및 오버클라우드 배포 스크립트를 다시 실행합니다. 예를 들면 다음과 같습니다.

1. **IdM과의 통합을 위해 python-novajoin 을 설치합니다.**

```
[stack@undercloud-0 ~]$ sudo yum install python-novajoin
```

2. **novajoin 구성 스크립트를 실행하고 IdM 배포에 대한 구성 세부 정보를 제공합니다. 예를 들면 다음과 같습니다.**

```
[stack@undercloud-0 ~]$ sudo novajoin-ipa-setup --principal admin --password
ComplexRedactedPassword \
  --server ipa.lab.local --realm lab.local --domain lab.local \
  --hostname undercloud-0.lab.local --precreate
...
0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
```

이 출력에는 IdM용 OTP(한 번 암호)가 포함되어 있습니다. 이 값은 배포에 다른 값이 됩니다.

3. **novajoin 을 사용하도록 Undercloud를 구성하고, 일회성 암호(OTP)를 추가하고, DNS에 IdM IP 주소를 사용하고, Overcloud 도메인을 설명합니다. 배포를 위해 이 예제를 조정해야 합니다.**

```
[stack@undercloud ~]$ vi undercloud.conf
...
enable_novajoin = true
ipa_otp = 0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
undercloud_hostname = undercloud-0.lab.local
undercloud_nameservers = X.X.X.X
overcloud_domain_name = lab.local
...
```

4. **언더클라우드에 novajoin 서비스를 설치합니다.**

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

5. **Overcloud IP 주소를 DNS에 추가합니다. 배포에 맞게 이 예제를 수정해야 합니다.**

참고: 오버클라우드의 network-environment.yaml 을 확인하고 각 네트워크 범위 내에서

VIP를 선택합니다.

```
[root@ipa ~]$ ipa dnsrecord-add lab.local overcloud --a-rec=10.0.0.101
[root@ipa ~]# ipa dnszone-add ctlplane.lab.local
[root@ipa ~]# ipa dnsrecord-add ctlplane.lab.local overcloud --a-rec 192.168.24.101
[root@ipa ~]# ipa dnszone-add internalapi.lab.local
[root@ipa ~]# ipa dnsrecord-add internalapi.lab.local overcloud --a-rec 172.17.1.101
[root@ipa ~]# ipa dnszone-add storage.lab.local
[root@ipa ~]# ipa dnsrecord-add storage.lab.local overcloud --a-rec 172.17.3.101
[root@ipa ~]# ipa dnszone-add storagemgmt.lab.local
[root@ipa ~]# ipa dnsrecord-add storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

6. 모든 끝점에 대한 **public_vip.yaml** 매핑을 생성합니다.

```
Parameter_defaults:
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.17.1.101'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.17.3.101'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.17.4.101'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.17.1.102'}]
```

17.4.2. IdM과 동일한 도메인을 사용하고 기존 공용 끝점 인증서를 유지하는 배포에 대해 오버클라우드 통합 구성

1. **openstack overcloud deploy** 명령(유효한 설정 포함)에 다음 매개변수가 있는지 확인한 다음 배포 명령을 다시 실행합니다.

- **' --NTP-server' -** 아직 설정되지 않은 경우 환경에 맞게 **NTP** 서버를 지정합니다. **IdM** 서버가 **ntp**를 실행해야 합니다.
- **cloud-names.yaml -** 초기 배포 명령에서 **FQDN(IP** 아님)을 포함합니다.
- **enable-tls.yaml -** 새 오버클라우드 인증서가 포함되어 있습니다. 예를 들어 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml 참조하십시오.
- **public_vip.yaml - dns**가 일치하도록 엔드포인트를 특정 **ip**에 매핑합니다.
- **enable-internal-tls.yaml -** 내부 엔드포인트에 **TLS**를 활성화합니다.

- **tls-everywhere-endpoints-dns.yaml - DNS** 이름을 사용하여 **TLS** 엔드포인트를 구성합니다. 이 파일의 내용을 검토하여 구성 범위를 확인할 수 있습니다.

- **haproxy-internal-tls-certmonger.yaml - certmonger**는 **haproxy**의 내부 인증서를 관리합니다.

- **inject-trust-anchor.yaml -** 루트 인증 기관을 추가합니다. 인증서가 기본적으로 사용되는 일반 세트의 일부가 아닌 **CA** 체인을 사용하는 경우에만 필요합니다(예: 자체 서명 사용).

  예를 들면 다음과 같습니다.

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-internal-tls-certmonger.yaml
\
  -e /home/stack/inject-trust-anchor.yaml
...
```

참고

이러한 환경 파일의 예는 https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl 입니다.

17.4.3. IdM과 동일한 도메인을 사용하는 배포용 오버클라우드 통합 구성 및 기존 공용 끝점 인증서를 IdM 생성 인증서로 교체

1. **openstack overcloud deploy** 명령(유효한 설정 포함)에 다음 매개변수가 있는지 확인한 다음 배포 명령을 다시 실행합니다.

   - **' --NTP-server' -** 아직 설정되지 않은 경우 환경에 맞게 **NTP** 서버를 지정합니다. **IdM** 서버가 **ntp**를 실행해야 합니다.

   - **cloud-names.yaml -** 초기 배포 명령에서 **FQDN(IP** 아님)을 포함합니다.

- **enable-tls.yaml -** 새 오버클라우드 인증서가 포함되어 있습니다. 예를 들어 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml 참조하십시오.

- **public_vip.yaml - dns**가 일치하도록 엔드포인트를 특정 **ip**에 매핑합니다.

- **enable-internal-tls.yaml -** 내부 엔드포인트에 **TLS**를 활성화합니다.

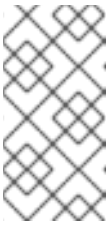
- **tls-everywhere-endpoints-dns.yaml - DNS** 이름을 사용하여 **TLS** 엔드포인트를 구성합니다. 이 파일의 내용을 검토하여 구성 범위를 확인할 수 있습니다.

- **haproxy-public-tls-certmonger.yaml - certmonger**는 **haproxy**에서 내부 및 공용 인증서를 관리합니다.

- **inject-trust-anchor.yaml -** 루트 인증 기관을 추가합니다. 인증서가 기본적으로 사용되는 일반 세트의 일부가 아닌 **CA** 체인을 사용하는 경우에만 필요합니다(예: 자체 서명 사용).

  예를 들면 다음과 같습니다.

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-public-tls-certmonger.yaml \
  -e /home/stack/inject-trust-anchor.yaml
...
```



참고

이러한 환경 파일의 예는 https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl 에서 찾을 수 있습니다.

참고

템플릿 **enable-internal-tls.j2.yaml** 은 오버클라우드 배포 명령에서 **enable-internal-tls.yaml** 로 참조됩니다.

또한 **enable-tls.yaml** 의 이전 공용 엔드포인트 인증서는 **certmonger**로 **haproxy-public-tls-certmonger.yaml** 로 교체되지만 업그레이드 프로세스에서 이 파일을 참조해야 합니다.

17.4.4. IdM 하위 도메인을 사용하는 배포를 위해 언더클라우드 통합 구성

다음 절차에서는 **IdM** 하위 도메인을 사용하는 배포를 위해 언더클라우드 통합을 구성하는 방법을 설명합니다.

Red Hat OpenStack Platform은 **novajoin** 을 사용하여 **Red Hat IdM(Identity Management)**과 통합되어 암호화 인증서를 발급하고 관리합니다. 이 절차에서는 언더클라우드를 **IdM**에 등록하고, 토큰을 생성하고, 언더클라우드 설정에서 토큰을 활성화한 다음, 언더클라우드 및 오버클라우드 배포 스크립트를 다시 실행합니다. 예를 들면 다음과 같습니다.

1. **IdM**과의 통합을 위해 **python-novajoin** 을 설치합니다.

```
[stack@undercloud-0 ~]$
```

2. **novajoin** 구성 스크립트를 실행하고 **IdM** 배포에 대한 구성 세부 정보를 제공합니다. 예를 들면 다음과 같습니다.

```
[stack@undercloud-0 ~]$ sudo novajoin-ipa-setup --principal admin --password
ComplexRedactedPassword \
  --server ipa.lab.local --realm lab.local --domain lab.local \
  --hostname undercloud-0.site1.lab.local --precreate
...
0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
```

이 출력에는 **IdM**용 **OTP**(한 번 암호)가 포함되어 있습니다. 이 값은 배포에 다른 값이 됩니다.

3. **novajoin** 을 사용하도록 언더클라우드를 구성하고 **DNS** 및 **NTP**용 **OTP, IdM IP** 및 오버클라우드 도메인을 추가합니다.

```
[stack@undercloud ~]$ vi undercloud.conf
```

```
…
[DEFAULT]
undercloud_ntp_servers=X.X.X.X
hieradata_override = /home/stack/hiera_override.yaml
enable_novajoin = true
ipa_otp = 0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
undercloud_hostname = undercloud-0.site1.lab.local
undercloud_nameservers = X.X.X.X
overcloud_domain_name = site1.lab.local
...
```

4. **novajoin** 을 사용하도록 언더클라우드를 구성하고 **DNS** 및 오버클라우드 도메인에 **OTP, IdM IP**를 추가합니다.

```
[stack@undercloud-0 ~]$ vi hiera_override.yaml
nova::metadata::novajoin::api::ipa_domain: site1.lab.local
...
```

5. 언더클라우드에 **novajoin** 서비스를 설치합니다.

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

6. **Overcloud IP** 주소를 **DNS**에 추가합니다. 배포에 맞게 이 예제를 수정해야 합니다.

참고**:** 오버클라우드의 **network-environment.yaml** 을 확인하고 각 네트워크 범위 내에서 **VIP**를 선택합니다.

```
[root@ipa ~]$ ipa dnsrecord-add site1.lab.local overcloud --a-rec=10.0.0.101
[root@ipa ~]# ipa dnszone-add site1.ctlplane.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.ctlplane.lab.local overcloud --a-rec 192.168.24.101
[root@ipa ~]# ipa dnszone-add site1.internalapi.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.internalapi.lab.local overcloud --a-rec 172.17.1.101
[root@ipa ~]# ipa dnszone-add site1.storage.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.storage.lab.local overcloud --a-rec 172.17.3.101
[root@ipa ~]# ipa dnszone-add site1.storagemgmt.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

7. 각 엔드포인트에 대한 **public_vip.yaml** 매핑을 생성합니다. 예를 들면 다음과 같습니다.

```
Parameter_defaults:
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.17.1.101'}]
```

```
  StorageVirtualFixedIPs: [{'ip_address':'172.17.3.101'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.17.4.101'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.17.1.102'}]
```

8. 각 엔드포인트에 대해 **extras.yaml** 매핑을 생성합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  MakeHomeDir: True
  IdMNoNtpSetup: false
  IdMDomain: redhat.local
  DnsSearchDomains: ["site1.redhat.local","redhat.local"]
```

17.4.5. IdM 하위 도메인을 사용하고 기존 공용 끝점 인증서를 유지하는 배포에 언더클라우드 통합 구성

다음 절차에서는 **IdM** 하위 도메인을 사용하고 기존 공용 엔드포인트 인증서를 유지하는 배포를 위해 **Undercloud** 통합을 구성하는 방법을 설명합니다.

1. **openstack overcloud deploy** 명령(유효한 설정 포함)에 다음 매개변수가 있는지 확인한 다음 배포 명령을 다시 실행합니다.

- **' --NTP-server' -** 아직 설정되지 않은 경우 환경에 맞게 **NTP** 서버를 지정합니다. **IdM** 서버가 **ntp**를 실행해야 합니다.
- **cloud-names.yaml -** 초기 배포 명령에서 **FQDN(IP** 아님)을 포함합니다.
- **enable-tls.yaml -** 새 오버클라우드 인증서가 포함되어 있습니다. 예를 들어 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml 참조하십시오.
- **public_vip.yaml - dns**가 일치하도록 엔드포인트가 특정 **ip**에 매핑됩니다.
- **'extras.yaml ' -** 로그인 시 홈 디렉토리를 만들고, **ntp** 설정, 기본 **IdM** 도메인 및 **dns**에서 **resolv.conf**를 검색하는 설정이 포함되어 있습니다.
- **enable-internal-tls.yaml -** 내부 엔드포인트에 **TLS**를 활성화합니다.
- **tls-everywhere-endpoints-dns.yaml - DNS** 이름을 사용하여 **TLS** 엔드포인트를 구성

합니다. 이 파일의 내용을 검토하여 구성 범위를 확인할 수 있습니다.

- **haproxy-internal-tls-certmonger.yaml - certmonger**는 **haproxy**의 내부 인증서를 관리합니다.

- **inject-trust-anchor.yaml -** 루트 인증 기관을 추가합니다. 인증서가 기본적으로 사용되는 일반 세트의 일부가 아닌 **CA** 체인을 사용하는 경우에만 필요합니다(예: 자체 서명 사용).

예를 들면 다음과 같습니다.

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e /home/stack/extras.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-internal-tls-certmonger.yaml
\
  -e /home/stack/inject-trust-anchor.yaml
...
```

참고

이러한 환경 파일의 예는 https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl 입니다.

17.4.6. IdM 하위 도메인을 사용하는 배포에 언더클라우드 통합 구성 및 기존 공용 끝점 인증서를 IdM 생성 인증서로 교체

다음 절차에서는 **IdM** 하위 도메인을 사용하는 배포에 언더클라우드 통합을 구성하는 방법과 기존 공용 엔드포인트 인증서를 **IdM** 생성 인증서로 바꾸는 방법을 설명합니다.

1. **openstack overcloud deploy** 명령(유효한 설정 포함)에 다음 매개변수가 있는지 확인한 다음 배포 명령을 다시 실행합니다.

- **' --NTP-server' -** 아직 설정되지 않은 경우 환경에 맞게 **NTP** 서버를 지정합니다. **IdM** 서버가 **ntp**를 실행해야 합니다.

- **cloud-names.yaml -** 초기 배포 명령에서 **FQDN(IP** 아님)을 포함합니다.

- **enable-tls.yaml -** 새 오버클라우드 인증서가 포함되어 있습니다. 예를 들어 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml 참조하십시오.

- **public_vip.yaml - dns**가 일치하도록 엔드포인트를 특정 **ip**에 매핑합니다.

- **'extras.yaml ' -** 로그인 시 홈 디렉토리를 만들고**, ntp** 설정**,** 기본 **ldM** 도메인 및 **dns**에서 **resolv.conf**를 검색하는 설정이 포함되어 있습니다.

- **enable-internal-tls.yaml -** 내부 엔드포인트에 **TLS**를 활성화합니다.

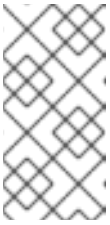
- **tls-everywhere-endpoints-dns.yaml - DNS** 이름을 사용하여 **TLS** 엔드포인트를 구성합니다. 이 파일의 내용을 검토하여 구성 범위를 확인할 수 있습니다.

- **haproxy-public-tls-certmonger.yaml - certmonger**는 **haproxy**에서 내부 및 공용 인증서를 관리합니다.

- **inject-trust-anchor.yaml -** 루트 인증 기관을 추가합니다. 인증서가 기본적으로 사용되는 일반 세트의 일부가 아닌 **CA** 체인을 사용하는 경우에만 필요합니다**(**예**:** 자체 서명 사용**).**

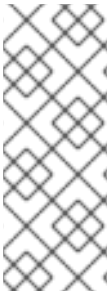
  예를 들면 다음과 같습니다.

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e /home/stack/extras.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-public-tls-certmonger.yaml \
  -e /home/stack/inject-trust-anchor.yaml
...
```



참고

이러한 환경 파일의 예는 https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl 입니다.



참고

이 예에서 **enable-internal-tls.j2.yaml** 템플릿은 오버클라우드 **deploy** 명령에서 **enable-internal-tls.yaml** 로 참조됩니다. 또한 **enable-tls.yaml** 의 이전 공용 엔드포인트 인증서는 **haproxy-public-tls-certmonger.yaml**을 사용하여 **certmonger** 로 교체되지만 이 파일은 업그레이드 프로세스에서 계속 참조해야 합니다.

17.5. TLS 암호화 확인

오버클라우드 재배포가 완료되면 모든 엔드포인트가 이제 **TLS**로 암호화되었는지 확인합니다. 이 예제에서 모든 끝점은 **TLS** 암호화를 사용하고 있음을 나타내는 **https** 를 사용하도록 구성됩니다.

```
+----------------------------------+-----------+-------------+---------------+---------+-----------+----------------
----------------------------------------------+
| ID                               | Region    | Service Name | Service Type   | Enabled | Interface | URL
|
+----------------------------------+-----------+-------------+---------------+---------+-----------+----------------
----------------------------------------------+
| 0fee4efdc4ae4310b6a139a25d9c0d9c | regionOne | neutron      | network       | True    | public    |
https://overcloud.lab.local:13696                |
| 220558ab1d2445139952425961a0c89a | regionOne | glance       | image         | True    | public    |
https://overcloud.lab.local:13292                |
| 24d966109ffa419da850da946f19c4ca | regionOne | placement    | placement     | True    | admin     |
https://overcloud.internalapi.lab.local:8778/placement        |
| 27ac9e0d22804ee5bd3cd8c0323db49c | regionOne | nova         | compute       | True    | internal  |
https://overcloud.internalapi.lab.local:8774/v2.1             |
| 31d376853bd241c2ba1a27912fc896c6 | regionOne | swift        | object-store  | True    | admin     |
https://overcloud.storage.lab.local:8080                 |
| 350806234c784332bfb8615e721057e3 | regionOne | nova         | compute       | True    | admin     |
https://overcloud.internalapi.lab.local:8774/v2.1             |
| 49c312f4db6748429d27c60164779302 | regionOne | keystone     | identity      | True    | public    |
https://overcloud.lab.local:13000                |
| 4e535265c35e486e97bb5a8bc77708b6 | regionOne | nova         | compute       | True    | public    |
https://overcloud.lab.local:13774/v2.1                   |
| 5e93dd46b45f40fe8d91d3a5d6e847d3 | regionOne | keystone     | identity      | True    | admin     |
https://overcloud.ctlplane.lab.local:35357               |
| 6561984a90c742a988bf3d0acf80d1b6 | regionOne | swift        | object-store  | True    | public    |
https://overcloud.lab.local:13808/v1/AUTH_%(tenant_id)s       |
| 76b8aad0bdda4313a02e4342e6a19fd6 | regionOne | placement    | placement     | True    | public
| https://overcloud.lab.local:13778/placement              |
| 96b004d5217c4d87a38cb780607bf9fb | regionOne | placement    | placement     | True    | internal  |
https://overcloud.internalapi.lab.local:8778/placement        |
| 98489b4b107f4da596262b712c3fe883 | regionOne | glance       | image         | True    | internal  |
https://overcloud.internalapi.lab.local:9292             |
```

```
| bb7ab36f30b14b549178ef06ec74ff84 | regionOne | glance       | image          | True    | admin     |
https://overcloud.internalapi.lab.local:9292                  |
| c1547f7bf9a14e9e85eaaaeea26413b7 | regionOne | neutron      | network        | True    | admin     |
https://overcloud.internalapi.lab.local:9696                  |
| ca66f499ec544f42838eb78a515d9f1e | regionOne | keystone     | identity       | True    | internal  |
https://overcloud.internalapi.lab.local:5000                  |
| df0181358c07431390bc66822176281d | regionOne | swift        | object-store   | True    | internal  |
https://overcloud.storage.lab.local:8080/v1/AUTH_%(tenant_id)s |
| e420350ef856460991c3edbfbae917c1 | regionOne | neutron      | network        | True    | internal  |
https://overcloud.internalapi.lab.local:9696                  |
+----------------------------------+-----------+--------------+----------------+---------+-----------+----------------
-----------------------------------------------+
```

18장. 디버그 모드

오버클라우드의 특정 서비스에 대해 **DEBUG** 수준 로깅 모드를 활성화하고 비활성화할 수 있습니다. 서비스에 대한 디버그 모드를 구성하려면 각 디버그 매개 변수를 설정합니다.

예를 들어 **OpenStack Identity(keystone)**는 **KeystoneDebug** 매개 변수를 사용합니다. **debug.yaml** 환경 파일을 생성하여 디버그 매개 변수를 저장하고 **parameter_defaults** 섹션에 **KeystoneDebug** 매개 변수를 설정합니다.

```
parameter_defaults:
  KeystoneDebug: True
```

KeystoneDebug 매개변수를 **True** 로 설정하면 **/var/log/containers/keystone/keystone.log** 표준 **keystone** 로그 파일이 **DEBUG** 수준 로그를 사용하여 업데이트됩니다.

디버그 매개변수의 전체 목록은 ***Overcloud Parameters*** *가이드의* ***"Debug Parameters"*** *를 참조하십시오.*

19장. 스토리지 설정

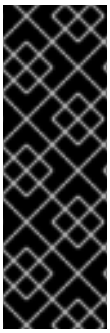
이 장에서는 오버클라우드의 스토리지 옵션을 구성하는 몇 가지 방법에 대해 간단히 설명합니다.

중요

기본적으로 오버클라우드는 **OpenStack Storage(cinder)**에서 제공하는 **OpenStack Compute(nova)** 및 **LVM** 블록 스토리지에서 제공하는 로컬 임시 스토리지를 사용합니다. 그러나 이러한 옵션은 엔터프라이즈급 오버클라우드에서는 지원되지 않습니다. 대신 이 장의 스토리지 옵션 중 하나를 사용합니다.

19.1. NFS 스토리지 구성

이 섹션에서는 **NFS** 공유를 사용하도록 오버클라우드를 구성하는 방법에 대해 설명합니다. 설치 및 구성 프로세스는 코어 **heat** 템플릿 컬렉션의 기존 환경 파일 수정을 기반으로 합니다.



중요

Red Hat은 인증된 스토리지 백엔드 및 드라이버를 사용하는 것이 좋습니다. 이 기능은 인증된 스토리지 백엔드 및 드라이버에 비해 기능이 제한되기 때문에 일반 **NFS** 백엔드에서 제공되는 **NFS**를 사용하지 않는 것이 좋습니다. 예를 들어 일반 **NFS** 백엔드는 볼륨 암호화 및 볼륨 다중 연결과 같은 기능을 지원하지 않습니다. 지원되는 드라이버에 대한 자세한 내용은 **Red Hat Ecosystem Catalog** 를 참조하십시오.

참고

NFS 백엔드 또는 **NetApp NFS** 블록 스토리지 백엔드에서 **NAS** 보안이라는 **NetApp** 기능을 지원하는지 여부를 제어하는 여러 **director heat** 매개변수가 있습니다.

- **CinderNetappNasSecureFileOperations**
- **CinderNetappNasSecureFilePermissions**
- **CinderNasSecureFileOperations**
- **CinderNasSecureFilePermissions**

Red Hat에서는 정상적인 볼륨 작업을 방해하기 때문에 이 기능을 활성화하지 않는 것이 좋습니다. **director**는 기본적으로 이 기능을 비활성화하고 **Red Hat OpenStack Platform**에서는 해당 기능을 지원하지 않습니다.

참고

블록 스토리지 및 계산 서비스의 경우 **NFS** 버전 **4.0** 이상을 사용해야 합니다.

코어 **heat** 템플릿 컬렉션에는 **/usr/share/openstack-tripleo-heat-templates/environments/** 에 일련의 환경 파일이 포함되어 있습니다. 이러한 환경 파일을 사용하면 **director**가 생성한 오버클라우드에서 지원되는 일부 기능에 대한 사용자 지정 구성을 생성할 수 있습니다. 여기에는 스토리지를 구성하도록 설계된 환경 파일이 포함됩니다. 이 파일은 **/usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml** 에 있습니다.

1. **stack** 사용자의 템플릿 디렉터리에 파일을 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml
~/templates/.
```

2. 다음 매개변수를 수정합니다.

CinderEnableIscsiBackend

iSCSI 백엔드를 활성화합니다. **false**로 설정합니다.

CinderEnableRbdBackend

Ceph Storage 백엔드를 활성화합니다. **false**로 설정합니다.

CinderEnableNfsBackend

NFS 백엔드를 활성화합니다. **true** 로 설정합니다.

NovaEnableRbdBackend

Nova 임시 스토리지를 위한 **Ceph Storage**를 활성화합니다. **false**로 설정합니다.

GlanceBackend

Glance에 사용할 백엔드를 정의합니다. 이미지에 파일 기반 스토리지를 사용하려면 파일로 설정합니다. 오버클라우드는 **Glance**를 위해 마운트된 **NFS** 공유에 이러한 파일을 저장합니다.

CinderNfsMountOptions

볼륨 스토리지의 **NFS** 마운트 옵션입니다.

CinderNfsServers

볼륨 스토리지에 마운트할 **NFS** 공유입니다. 예를 들면 **192.168.122.1:/export/cinder**입니다.

GlanceNfsEnabled

GlanceBackend 를 **file** 로 설정하면**GlanceNfsEnabled** 를 사용하면 모든 컨트롤러 노드가 이미지에 액세스할 수 있도록 공유 위치에 **NFS**를 통해 이미지를 저장할 수 있습니다. 비활성화된 경우 오버클라우드는 컨트롤러 노드의 파일 시스템에 이미지를 저장합니다. **true** 로 설정합니다.

GlanceNfsShare

이미지 스토리지용으로 마운트할 **NFS** 공유입니다. 예를 들면 **192.168.122.1:/export/glance**입니다.

GlanceNfsOptions

이미지 스토리지의 **NFS** 마운트 옵션입니다.

환경 파일에는 **Red Hat OpenStack Platform** 블록 스토리지**(cinder)** 및 이미지**(glance)** 서비스에 대한 다양한 스토리지 옵션을 구성하는 매개변수가 포함되어 있습니다. 다음 예제에

서는 **NFS** 공유를 사용하도록 오버클라우드를 구성하는 방법을 보여줍니다.

환경 파일의 옵션은 다음과 유사해야 합니다.

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
  GlanceBackend: file

  CinderNfsMountOptions: rw,sync,context=system_u:object_r:cinder_var_lib_t:s0
  CinderNfsServers: 192.0.2.230:/cinder

  GlanceNfsEnabled: true
  GlanceNfsShare: 192.0.2.230:/glance
  GlanceNfsOptions: rw,sync,context=system_u:object_r:glance_var_lib_t:s0
```

이러한 매개변수는 **heat** 템플릿 컬렉션의 일부로 통합됩니다. 예제 코드에 표시된 대로 설정하면 블록 스토리지 및 이미지 서비스에서 사용할 두 개의 **NFS** 마운트 지점이 생성됩니다.

중요

이미지 서비스가 **/ var/lib** 디렉터리에 액세스할 수 있도록 **context=system_u:object_r:glance_var _lib_t:s0** 옵션을 **GlanceNfsOptions** 매개 변수에 포함합니다. 이 **SELinux** 콘텐츠가 없으면 이미지 서비스에서 마운트 지점에 쓸 수 없습니다.

3. 오버클라우드를 배포할 때 파일을 포함합니다.

19.2. CEPH STORAGE 구성

director는 **Red Hat Ceph Storage**를 **Overcloud**에 통합하는 두 가지 기본 방법을 제공합니다.

자체 Ceph Storage 클러스터를 사용하여 오버클라우드 생성

director는 **Overcloud**를 생성하는 동안 **Ceph Storage** 클러스터를 생성할 수 있습니다. **director**는 **Ceph OSD**를 사용하여 데이터를 저장하는 **Ceph Storage** 노드 집합을 생성합니다. 또한 **director**는 **Overcloud**의 컨트롤러 노드에 **Ceph Monitor** 서비스를 설치합니다. 즉, 조직이 세 개의 고가용성

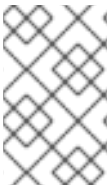
컨트롤러 노드를 사용하여 **Overcloud**를 생성하면 **Ceph** 모니터도 고가용성 서비스가 됩니다. 자세한 내용은 **Deploying an Overcloud with Containerized Red Hat Ceph** 가이드를 참조하십시오.

기존 **Ceph** 스토리지를 오버클라우드에 통합

기존 **Ceph Storage** 클러스터가 이미 있는 경우 오버클라우드 배포 중에 이를 통합할 수 있습니다. 즉, **Overcloud** 구성 외부에서 클러스터를 관리하고 확장할 수 있습니다. 자세한 내용은 **Integrating an Overcloud with an Existing Red Hat Ceph Cluster** 가이드를 참조하십시오.

19.3. 외부 오브젝트 스토리지 클러스터 사용

컨트롤러 노드에서 기본 **Object Storage** 서비스 배포를 비활성화하여 외부 **Object Storage(swift)** 클러스터를 재사용할 수 있습니다. 이렇게 하면 오브젝트 스토리지의 프록시 및 스토리지 서비스를 모두 비활성화하고 지정된 외부 **Swift** 엔드포인트를 사용하도록 **haproxy** 및 **keystone**을 구성합니다.



참고

외부 **Object Storage(swift)** 클러스터의 사용자 계정은 직접 관리해야 합니다.

외부 **Object Storage** 클러스터의 엔드포인트 **IP** 주소와 외부 **Object Storage proxy-server.conf** 파일의 **authtoken** 암호가 필요합니다. **openstack endpoint list** 명령을 사용하여 이 정보를 찾을 수 있습니다.

외부 **Swift** 클러스터가 있는 **director**를 배포하려면 다음을 수행합니다.

1. 다음 콘텐츠를 사용하여 **swift-external-params.yaml** 이라는 새 파일을 생성합니다.

   - **EXTERNAL.IP:PORT** 를 외부 프록시의 **IP** 주소 및 포트로 바꿉니다.
   - **AUTHTOKEN** 을 **SwiftPassword** 행의 외부 프록시의 **authtoken** 암호로 교체합니다.

```
parameter_defaults:
  ExternalPublicUrl: 'https://EXTERNAL.IP:PORT/v1/AUTH_%(tenant_id)s'
  ExternalInternalUrl: 'http://192.168.24.9:8080/v1/AUTH_%(tenant_id)s'
  ExternalAdminUrl: 'http://192.168.24.9:8080'
  ExternalSwiftUserTenant: 'service'
  SwiftPassword: AUTHTOKEN
```

2. 이 파일을 **swift-external-params.yaml** 로 저장합니다.

3. 이러한 추가 환경 파일을 사용하여 **Overcloud**를 배포합니다.

```
openstack overcloud deploy --templates \
-e [your environment files]
-e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml
-e swift-external-params.yaml
```

19.4. 이미지 가져오기 방법 및 공유 스테이징 영역 구성

OpenStack Image 서비스(**glance**)의 기본 설정은 **OpenStack** 설치 시 사용되는 **Heat** 템플릿에 의해 결정됩니다. 이미지 서비스 **Heat** 템플릿은 **tht/puppet/services/glance-api.yaml** 입니다.

상호 운용 가능한 이미지 가져오기를 사용하면 두 가지 이미지 가져오기 방법이 있습니다.

- **web-download** 및
- **glance-direct.**

web-download 방법을 사용하면 **URL**에서 이미지를 가져올 수 있습니다. **glance-direct** 방법을 사용하면 로컬 볼륨에서 이미지를 가져올 수 있습니다.

19.4.1. glance-settings.yaml 파일 생성 및 배포

환경 파일을 사용하여 가져오기 매개 변수를 구성합니다. 이러한 매개변수는 **Heat** 템플릿에 설정된 기본값을 재정의합니다. 예제 환경 콘텐츠는 상호 운용 가능한 이미지 가져오기에 대한 매개 변수를 제공합니다.

```
parameter_defaults:
  # Configure NFS backend
  GlanceBackend: file
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.168.122.1:/export/glance

  # Enable glance-direct import method
  GlanceEnabledImportMethods: glance-direct,web-download
```

```
# Configure NFS staging area (required for glance-direct import method)
GlanceStagingNfsShare: 192.168.122.1:/export/glance-staging
```

GlanceBackend,GlanceNfsEnabled 및 **GlanceNfsShare** 매개 변수는 ***Advanced Overcloud Customization Guide*** 의 **Storage Configuration(스토리지 구성)** 섹션에 정의되어 있습니다.

상호 운용 가능한 이미지 가져오기를 위한 두 개의 새 매개 변수는 가져오기 방법과 공유 **NFS** 스테이징 영역을 정의합니다.

GlanceEnabledImportMethods

사용 가능한 가져오기 방법, **web-download(기본값)** 및 **glance-direct**를 정의합니다. 이 행은 **web-download** 이외의 추가 방법을 활성화하려는 경우에만 필요합니다.

GlanceStagingNfsShare

glance-direct 가져오기 방법에서 사용하는 **NFS** 스테이징 영역을 구성합니다. 이 공간은 고가용성 클러스터 설정의 노드 간에 공유할 수 있습니다. **GlanceNfsEnabled**를 **true**로 설정해야 합니다.

설정을 구성하려면 다음을 수행합니다.

1. 라는 새 파일을 만듭니다(예: **glance-settings.yaml**). 이 파일의 내용은 위의 예제와 유사해야 합니다.

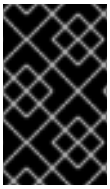
2. **openstack overcloud deploy** 명령을 사용하여 파일을 **OpenStack** 환경에 추가합니다.

```
$ openstack overcloud deploy --templates -e glance-settings.yaml
```

환경 파일 사용에 대한 자세한 내용은 ***Advanced Overcloud Customization Guide 의 Overcloud Creation에서 환경 파일 포함 섹션을 참조하십시오.***

19.5. 이미지 서비스의 CINDER 백엔드 구성

GlanceBackend 매개 변수는 이미지 서비스에서 이미지를 저장하는 데 사용하는 백엔드를 설정합니다.



중요

프로젝트에 대해 생성할 수 있는 기본 최대 볼륨 수는 **10**입니다.

절차

1. **cinder** 를 이미지 서비스 백엔드로 구성하려면 환경 파일에 다음을 추가합니다.

```
parameter_defaults:
  GlanceBackend: cinder
```

2. **cinder** 백엔드가 활성화된 경우 기본적으로 다음 매개변수와 값이 설정됩니다.

```
cinder_store_auth_address = http://172.17.1.19:5000/v3
cinder_store_project_name = service
cinder_store_user_name = glance
cinder_store_password = ****secret****
```

3. 사용자 정의 사용자 이름 또는 **cinder_store_** 매개변수의 사용자 지정 값을 사용하려면 **ExtraConfig** 설정을 **parameter _defaults** 에 추가하고 사용자 정의 값을 전달합니다.

```
ExtraConfig:
    glance::config::api_config:
      glance_store/cinder_store_auth_address:
        value: "%{hiera('glance::api::authtoken::auth_url')}/v3"
      glance_store/cinder_store_user_name:
        value: <user-name>
      glance_store/cinder_store_password:
        value: "%{hiera('glance::api::authtoken::password')}"
      glance_store/cinder_store_project_name:
        value: "%{hiera('glance::api::authtoken::project_name')}"
```

19.6. 하나의 인스턴스에 연결할 최대 스토리지 장치 수 구성

기본적으로 무제한 스토리지 장치를 단일 인스턴스에 연결할 수 있습니다. 최대 장치 수를 제한하려면 **max_disk_devices_to_attach** 매개 변수를 **Compute** 환경 파일에 추가합니다. 다음 예제에서는 **max_disk_devices_to_attach** 값을 **"30"**으로 변경하는 방법을 보여줍니다.

```
parameter_defaults:
  ComputeExtraConfig:
        nova::config::nova_config:
          compute/max_disk_devices_to_attach:
            value: '30'
```

지침 및 고려 사항

- 인스턴스에서 지원하는 스토리지 디스크 수는 디스크에서 사용하는 버스에 따라 다릅니다. 예를 들어 IDE 디스크 버스는 4개의 연결된 장치로 제한됩니다.
- 활성 인스턴스가 있는 컴퓨팅 노드에서 max_disk_devices_to_attach 를 변경하면 인스턴스에 이미 연결된 장치 수보다 낮은 경우 최대 수가 다시 빌드되지 않을 수 있습니다. 예를 들어 A 인스턴스에 26개의 장치가 연결되어 있고 max_disk_devices_to_attach 를 20으로 변경하면 A 인스턴스를 다시 빌드하는 요청이 실패합니다.
- 콜드 마이그레이션 중에 구성된 최대 스토리지 장치 수는 마이그레이션하려는 인스턴스의 소스에만 적용됩니다. 이동하기 전에 대상을 확인하지 않습니다. 즉, 컴퓨팅 노드 A에 26개의 연결된 디스크 장치가 있고 컴퓨팅 노드 B에 최대 20개의 연결된 디스크 장치가 구성되어 있으면 컴퓨팅 노드 A에서 컴퓨팅 노드 B로 26개의 연결된 인스턴스를 콜드 마이그레이션합니다. 그러나 구성된 최대 20개를 초과하는 26개의 장치가 이미 연결되어 있으므로 컴퓨팅 노드 B에서 인스턴스를 다시 빌드하기 위한 후속 요청은 실패합니다.
- 컴퓨팅 노드가 없으므로 구성된 최대값은 보류된 오프로드된 인스턴스에 적용되지 않습니다.
- 인스턴스에 많은 디스크 장치를 연결하면 인스턴스의 성능이 저하될 수 있습니다. 환경에서 지원할 수 있는 경계에 따라 최대 개수를 조정해야 합니다.
- 시스템 유형 Q35가 있는 인스턴스는 최대 500개의 디스크 장치를 연결할 수 있습니다.

19.7. 타사 스토리지 구성

director에는 타사 스토리지 프로바이더를 구성하는 데 도움이 되는 몇 가지 환경 파일이 포함되어 있습니다. 여기에는 다음이 포함됩니다.

Dell EMC 스토리지 센터

블록 스토리지(cinder) 서비스를 위한 단일 Dell EMC Storage Center 백엔드 배포.

환경 파일은 /usr/share/openstack-tripleo-heat-templates/environments/cinder-dellsc-config.yaml 에 있습니다.

전체 구성 정보는 Dell 스토리지 센터 백엔드 가이드를 참조하십시오.

Dell EMC PS 시리즈

블록 스토리지(cinder) 서비스를 위한 단일 Dell EMC PS 시리즈 백엔드 배포.

환경 파일은 /usr/share/openstack-tripleo-heat-templates/environments/cinder-dellps-config.yaml 에 있습니다.

전체 구성 정보는 Dell EMC PS 시리즈 백엔드 가이드를 참조하십시오.

NetApp Block Storage

NetApp 스토리지 어플라이언스를 블록 스토리지(cinder) 서비스의 백엔드로 배포합니다.

환경 파일은 /usr/share/openstack-tripleo-heat-templates/environments/storage/cinder-netapp-config.yaml 에 있습니다.

전체 구성 정보는 NetApp 블록 스토리지 백엔드 가이드를 참조하십시오.

20장. 보안 개선 사항

다음 섹션에서는 오버클라우드의 보안을 강화하기 위한 몇 가지 제안 사항을 제공합니다.

20.1. 오버클라우드 방화벽 관리

각 핵심 OpenStack Platform 서비스에는 구성 가능 서비스 템플릿에 방화벽 규칙이 포함되어 있습니다. 이렇게 하면 각 오버클라우드 노드에 대한 기본 방화벽 규칙 세트가 자동으로 생성됩니다.

오버클라우드 Heat 템플릿에는 추가 방화벽 관리에 도움이 되는 매개변수 세트가 포함되어 있습니다.

ManageFirewall

방화벽 규칙을 자동으로 관리할지 여부를 정의합니다. Puppet에서 각 노드에서 방화벽을 자동으로 구성할 수 있도록 하려면 true 로 설정합니다. 방화벽을 수동으로 관리하려면 false 로 설정합니다. 기본값은 true입니다.

PurgeFirewallRules

새 Linux 방화벽 규칙을 구성하기 전에 기본 Linux 방화벽 규칙을 제거할지 여부를 정의합니다. 기본값은 false입니다.

ManageFirewall 을 true 로 설정하면 배포에 추가 방화벽 규칙을 생성할 수 있습니다. 오버클라우드의 환경 파일에서 구성 후크를 사용하여 tripleo::firewall::firewall_rules hieradata를 설정합니다(4.5절. “Puppet: 역할에 맞는 Hieradata 사용자 정의”참조). 이 hieradata는 방화벽 규칙 이름과 해당 매개 변수를 키로 포함하는 해시이며, 모두 선택 사항입니다.

port

규칙에 연결된 포트입니다.

dport

규칙에 연결된 대상 포트입니다.

sport

규칙과 연결된 소스 포트입니다.

Proto

규칙과 연결된 프로토콜입니다. 기본값은 tcp 입니다.

작업

규칙과 연결된 작업 정책입니다. 기본값은 **accept** 입니다.

건너뛰기

로 이동할 체인입니다. 있는 경우 작업을 재정의합니다.

상태

규칙과 관련된 상태 배열입니다. 기본값은 **['NEW']** 입니다.

소스

규칙과 연결된 소스 **IP** 주소입니다.

iniface

규칙에 연결된 네트워크 인터페이스입니다.

체인

규칙과 연결된 체인입니다. 기본값은 **INPUT** 입니다.

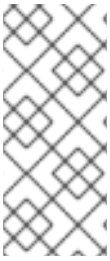
대상

규칙에 연결된 대상 **CIDR**입니다.

다음 예제에서는 방화벽 규칙 형식의 구문을 보여줍니다.

```
ExtraConfig:
  tripleo::firewall::firewall_rules:
    '300 allow custom application 1':
      port: 999
      proto: udp
      action: accept
    '301 allow custom application 2':
      port: 8081
      proto: tcp
      action: accept
```

이는 **ExtraConfig** 를 통해 모든 노드에 **2**개의 추가 방화벽 규칙을 적용합니다.



참고

각 규칙 이름은 해당 **iptables** 규칙의 주석이 됩니다. 또한 각 규칙 이름은 **3**자리 접두사로 시작되어 **Puppet**이 최종 **iptables** 파일에서 정의된 모든 규칙을 정렬하는 데 도움이 됩니다. 기본 **OpenStack Platform** 규칙은 **000~200** 범위의 접두사를 사용합니다.

20.2. SNMP(SIMPLE NETWORK MANAGEMENT PROTOCOL) 문자열 변경

director는 오버클라우드에 대한 기본 읽기 전용 **SNMP** 설정을 제공합니다. 네트워크 장치에 대해 학습하는 권한이 없는 사용자의 위험을 완화하도록 **SNMP** 문자열을 변경하는 것이 좋습니다.

오버클라우드의 환경 파일에서 **ExtraConfig** 후크를 사용하여 다음 **hieradata**를 설정합니다.

SNMP 기존 액세스 제어 설정

snmp::ro_community

IPv4 읽기 전용 **SNMP** 커뮤니티 문자열. 기본값은 **public** 입니다.

snmp::ro_community6

IPv6 읽기 전용 **SNMP** 커뮤니티 문자열. 기본값은 **public** 입니다.

snmp::ro_network

RO에서 데몬을 쿼리 할 수 있는 네트워크입니다. 이 값은 문자열 또는 배열일 수 있습니다. 기본값은 **127.0.0.1** 입니다.

snmp::ro_network6

RO가 **IPv6**로 데몬을 쿼리 할 수 있는 네트워크입니다. 이 값은 문자열 또는 배열일 수 있습니다. 기본값은 **::1/128** 입니다.

tripleo::profile::base::snmp::snmpd_config

안전 위로 the ***snmpd.conf*** 파일에 추가할 행 배열입니다. 기본값은 **[]** 입니다. 사용 가능한 모든 옵션은 **SNMP** 구성 파일 웹 페이지를 참조하십시오.

예를 들면 다음과 같습니다.

```
parameter_defaults:
```

```
ExtraConfig:
  snmp::ro_community: mysecurestring
  snmp::ro_community6: myv6securestring
```

그러면 모든 노드에서 읽기 전용 **SNMP** 커뮤니티 문자열이 변경됩니다.

SNMP 보기 기반 액세스 제어 설정(**VACM**)

snmp::com2sec

IPv4 보안 이름.

snmp::com2sec6

IPv6 보안 이름.

예를 들면 다음과 같습니다.

```
parameter_defaults:
  ExtraConfig:
    snmp::com2sec: mysecurestring
    snmp::com2sec6: myv6securestring
```

그러면 모든 노드에서 읽기 전용 **SNMP** 커뮤니티 문자열이 변경됩니다.

자세한 내용은 **the snmpd.conf** 도움말 페이지를 참조하십시오.

20.3. HAPROXY에 대한 SSL/TLS 암호화 방식 및 규칙 변경

오버클라우드에서 **SSL/TLS**를 활성화한 경우(**15**장. *오버클라우드 공개 엔드 포인트에서 SSL/TLS 활성화*참조) **HAProxy** 설정에 사용되는 **SSL/TLS** 암호화 방식 및 규칙을 강화할 수 있습니다. 이는 **POODLE** 취약점과 같은 **SSL/TLS** 취약점을 방지하는 데 도움이 됩니다.

오버클라우드의 환경 파일에서 **ExtraConfig** 후크를 사용하여 다음 **hieradata**를 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.

- **암호: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA -AES256-GCM-SHA384: ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256 :ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE- ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES128-SHA256:A ES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS**

- **규칙: no-sslv3 no-tls-tickets**

다음 콘텐츠를 사용하여 환경 파일을 생성합니다.

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
    tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

참고

암호화 방식 컬렉션은 연속된 한 행으로 되어 있습니다.

오버클라우드 생성을 통해 이 환경 파일을 포함합니다.

20.4. OPEN VSWITCH 방화벽 사용

Red Hat OpenStack Platform director에서 OVS(Open vSwitch) 방화벽 드라이버를 사용하도록 보안 그룹을 구성할 수 있습니다. NeutronOVSFirewallDriver 매개변수를 사용하면 사용할 방화벽 드라이버를 지정할 수 있습니다.

- iptables_hybrid - iptables/하이브리드 기반 구현을 사용하도록 neutron을 구성합니다.
- openvswitch - OVS 방화벽 흐름 기반 드라이버를 사용하도록 neutron을 구성합니다.

openvswitch 방화벽 드라이버는 성능이 향상되고 게스트를 프로젝트 네트워크에 연결하는 데 사용되는 인터페이스 및 브리지 수를 줄입니다.

중요

멀티캐스트 트래픽은 iptables 방화벽 드라이버와 OVS(Open vSwitch) 방화벽 드라이버에서 다르게 처리됩니다. 기본적으로 iptables를 사용하면 VRRP 트래픽이 거부되며 VRRP 트래픽이 끝점에 도달하기 위해 보안 그룹 규칙에서 VRRP를 활성화해야 합니다. OVS에서는 모든 포트가 동일한 OpenFlow 컨텍스트를 공유하며, 멀티캐스트 트래픽을 포트당 개별적으로 처리할 수 없습니다. 보안 그룹은 모든 포트(예: 라우터의 포트)에 적용되지 않으므로 OVS는 NORMAL 작업을 사용하고 RFC 4541에서 지정하는 모든 포트에 멀티캐스트 트래픽을 전달합니다.

참고

iptables_hybrid 옵션은 OVS-DPDK와 호환되지 않습니다.

network-environment.yaml 파일에서 NeutronOVSFirewallDriver 매개변수를 구성합니다.

```
NeutronOVSFirewallDriver: openvswitch
```

- NeutronOVSFirewallDriver : 보안 그룹을 구현할 때 사용할 방화벽 드라이버의 이름을 구성합니다. 가능한 값은 시스템 구성에 따라 다릅니다. 예를 들면 noop,openvswitch,iptables_hybrid 가 있습니다. 기본값은 iptables_hybrid 에 해당하는 빈 문자열입니다.

20.5. 보안 루트 사용자 액세스 사용

Overcloud 이미지에는 **root** 사용자의 강화된 보안이 자동으로 포함되어 있습니다. 예를 들어 배포된 각 **Overcloud** 노드는 **root** 사용자에 대한 직접 **SSH** 액세스를 자동으로 비활성화합니다. 다음 방법을 통해 오버클라우드 노드에서 **root** 사용자에게 계속 액세스할 수 있습니다.

1. **Undercloud** 노드의 **stack** 사용자에 로그인합니다.

2. 각 오버클라우드 노드에는 **heat-admin** 사용자 계정이 있습니다. 이 사용자 계정에는 언더클라우드의 공용 **SSH** 키가 포함되어 있으며, 언더클라우드에서 오버클라우드 노드로의 암호 없이 **SSH** 액세스를 제공합니다. 언더클라우드 노드에서 **heat-admin** 사용자를 사용하여 **SSH**를 통해 선택한 **Overcloud** 노드에 로그인합니다.

3. **sudo -i** 를 사용하여 **root** 사용자로 전환합니다.

루트 사용자 보안 감소

일부 상황에는 **root** 사용자에 대한 직접 **SSH** 액세스가 필요할 수 있습니다. 이 경우 각 오버클라우드 노드의 **root** 사용자에 대한 **SSH** 제한 사항을 줄일 수 있습니다.

주의

이 메서드는 디버깅 목적으로만 사용됩니다. 프로덕션 환경에서는 사용하지 않는 것이 좋습니다.

방법은 첫 번째 부팅 구성 후크를 사용합니다(**4.1**절. “첫 번째 부팅: 첫 번째 부팅 구성 사용자 정의”참조). 환경 파일에 다음 콘텐츠를 배치합니다.

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-
templates/firstboot/userdata_root_password.yaml

parameter_defaults:
  NodeRootPassword: "p@55w0rd!"
```

다음을 확인합니다.

- **OS::TripleO::NodeUserData** 리소스는 첫 번째 부팅 **cloud-init** 단계에서 **root** 사용자를 구성하는 템플릿을 참조합니다.

- **NodeRootPassword** 매개 변수는 **root** 사용자의 암호를 설정합니다. 이 매개 변수의 값을 원하는 암호로 변경합니다. 환경 파일에는 보안 위험으로 간주되는 일반 텍스트 문자열로 암호가 포함되어 있습니다.

오버클라우드 생성 시 **openstack overcloud deploy** 명령을 사용하여 이 환경 파일을 포함합니다.

21장. 네트워크 플러그인 구성

director에는 타사 네트워크 플러그인을 설정하는 데 도움이 되는 환경 파일이 포함되어 있습니다.

21.1. FUJITSU CONVERGED FABRIC (C-FABRIC)

/usr/share/openstack-tripleo-heat-templates/environments/neutron-✓2-fujitsu-cfabab.yaml에 있는 환경 파일을 사용하여 Fujitsu Converged Fabric(C- Fabric) 플러그인을 활성화할 수 있습니다.

1. 환경 파일을 templates 하위 디렉터리에 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml
/home/stack/templates/
```

2. 절대 경로를 사용하도록 resource_registry 를 편집합니다.

```
resource_registry:
  OS::TripleO::Services::NeutronML2FujitsuCfab: /usr/share/openstack-tripleo-heat-
templates/puppet/services/neutron-plugin-ml2-fujitsu-cfab.yaml
```

3. /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml에서 parameter_defaults 를 검토합니다.

- NeutronFujitsuCfabAddress - C-Fabric의 telnet IP 주소(문자열)
- NeutronFujitsuCfabUserName - 사용할 C-Fabric 사용자 이름(문자열)
- NeutronFujitsuCfabPassword - C-Fabric 사용자 계정의 암호(문자열)
- NeutronFujitsuCfabPhysicalNetworks - <physical_network>:<vfab_id> phyples로 physical_network 이름 및 해당 vfab ID를 지정합니다. (comma_delimited_list)
- NeutronFujitsuCfabSharePprofile - 동일한 VLAN ID를 사용하는 neutron 포트 간에 C-Fabric pprofile을 공유할지 여부를 결정합니다. (boolean)

- **NeutronFujitsuCfabPprofilePrefix - pprofile** 이름의 접두사 문자열(문자열)

- **NeutronFujitsuCfabsaveConfig -** 구성을 저장할지 여부를 결정합니다. (부울)

4. 배포에 템플릿을 적용하려면 **openstack overcloud deploy** 명령에 환경 파일을 포함합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml [OTHER OPTIONS] ...
```

21.2. FUJITSU FOS SWITCH

/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml 에 있는 환경 파일을 사용하여 **Fujitsu FOS Switch** 플러그인을 활성화할 수 있습니다.

1. 환경 파일을 **templates** 하위 디렉터리에 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml /home/stack/templates/
```

2. 절대 경로를 사용하도록 **resource_registry** 를 편집합니다.

```
resource_registry:
  OS::TripleO::Services::NeutronML2FujitsuFossw: /usr/share/openstack-tripleo-heat-templates/puppet/services/neutron-plugin-ml2-fujitsu-fossw.yaml
```

3. **/home/stack/templates/neutron-ml2-fujitsu-fossw.yaml**에서 **parameter_defaults** 를 검토합니다.

- **NeutronFujitsuFosswIps -** 모든 **FOS** 스위치의 **IP** 주소. **(comma_delimited_list)**

- **NeutronFujitsuFosswUserName -** 사용할 **FOS** 사용자 이름(문자열)

- **NeutronFujitsuFosswPassword - FOS** 사용자 계정의 암호(문자열)

- **NeutronFujitsuFosswPort - SSH** 연결에 사용할 포트 번호입니다(숫자)
- **NeutronFujitsuFosswTimeout - SSH** 연결의 시간 초과 기간(숫자)
- **NeutronFujitsuFosswUdpDestPort - FOS** 스위치의 **VXLAN UDP** 대상의 포트 번호입니다(숫자)
- **NeutronFujitsuFosswOvsdbVlanidRangeMin - VNI** 및 물리적 포트 바인딩에 사용되는 최소 **VLAN ID**입니다(숫자)
- **NeutronFujitsuFosswOvsdbPort - FOS** 스위치의 **OVSDB** 서버의 포트 번호입니다(숫자)

4. **배포에 템플릿을 적용하려면 openstack overcloud deploy 명령에 환경 파일을 포함합니다. 예를 들면 다음과 같습니다.**

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS] ...
```

22장. ID 구성

director에는 ID 서비스(keystone) 설정을 구성하는 데 도움이 되는 매개변수가 포함되어 있습니다.

22.1. 지역 이름

기본적으로 오버클라우드의 리전 이름은 **regionOne** 으로 지정됩니다. 환경 파일을 **KeystoneRegion** 항목을 추가하여 변경할 수 있습니다. 이 설정은 배포 후 변경할 수 없습니다.

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

23장. 기타 구성

23.1. 오버클라우드 노드에서 커널 구성

OpenStack Platform director에는 오버클라우드 노드에서 커널을 구성하는 매개변수가 포함되어 있습니다.

ExtraKernelModules

로드할 커널 모듈. 모듈 이름은 빈 값이 있는 해시 키로 나열됩니다.

```
ExtraKernelModules:
  <MODULE_NAME>: {}
```

ExtraKernelPackages

ExtraKernelModule에서 커널 모듈을 로드하기 전에 설치할 커널 관련 패키지입니다. 패키지 이름은 빈 값을 사용하여 해시 키로 나열됩니다.

```
ExtraKernelPackages:
  <PACKAGE_NAME>: {}
```

ExtraSysctlSettings

적용할 **sysctl** 설정 해시입니다. **value** 키를 사용하여 각 매개 변수의 값을 설정합니다.

```
ExtraSysctlSettings:
  <KERNEL_PARAMETER>:
    value: <VALUE>
```

이 예에서는 환경 파일에서 이러한 매개변수의 구문을 보여줍니다.

```
parameter_defaults:
  ExtraKernelModules:
    iscsi_target_mod: {}
  ExtraKernelPackages:
    iscsi-initiator-utils: {}
  ExtraSysctlSettings:
    dev.scsi.logging_level:
      value: 1
```

23.2. 외부 로드 밸런싱 구성

Overcloud는 여러 컨트롤러를 고가용성 클러스터로 함께 사용하여 OpenStack 서비스의 운영 성능을 극대화합니다. 또한 클러스터에서 OpenStack 서비스에 액세스할 수 있는 부하 분산을 제공하여 컨트롤러 노드에 트래픽을 균등하게 분산하고 각 노드의 서버 과부하를 줄입니다. 외부 로드 밸런서를 사용하여 이 배포를 수행할 수도 있습니다. 예를 들어 조직에서 자체 하드웨어 기반 로드 밸런서를 사용하여 컨트롤러 노드에 대한 트래픽 배포를 처리할 수 있습니다.

외부 로드 밸런싱 구성에 대한 자세한 내용은 전체 지침은 Overcloud 전용 External Load Balancing 가이드를 참조하십시오.

23.3. IPV6 네트워킹 구성

기본적으로 Overcloud는 IPv4)를 사용하여 서비스 엔드포인트를 구성합니다. 그러나 Overcloud는 IPv6 인프라를 지원하는 조직에 유용한 IPv6) 엔드포인트도 지원합니다. director에는 IPv6 기반 오버클라우드 생성에 도움이 되는 환경 파일 세트가 포함되어 있습니다.

Overcloud에서 IPv6 구성에 대한 자세한 내용은 전용 IPv6 네트워킹 for the Overcloud 가이드를 참조하십시오.