Red Hat OpenStack Platform 12

Advanced Overcloud Customization

Methods for configuring advanced features using Red Hat OpenStack Platform director

Last Updated: 2019-04-05

Red Hat OpenStack Platform 12 Advanced Overcloud Customization

Methods for configuring advanced features using Red Hat OpenStack Platform director

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide explains how to configure certain advanced features for a Red Hat OpenStack Platform enterprise environment using the Red Hat OpenStack Platform Director. This includes features such as network isolation, storage configuration, SSL communication, and general configuration methods.

Table of Contents

CHAPTER 1. INTRODUCTION 5

CHAPTER 2. UNDERSTANDING HEAT TEMPLATES 6
2.1. HEAT TEMPLATES 6
2.2. ENVIRONMENT FILES 7
2.3. CORE OVERCLOUD HEAT TEMPLATES 8
2.4. PLAN ENVIRONMENT METADATA 9
2.5. CAPABILITIES MAP 10
2.6. INCLUDING ENVIRONMENT FILES IN OVERCLOUD CREATION 11
2.7. USING CUSTOMIZED CORE HEAT TEMPLATES 12

CHAPTER 3. PARAMETERS 16
3.1. EXAMPLE 1: CONFIGURING THE TIMEZONE 16
3.2. EXAMPLE 2: DISABLING LAYER 3 HIGH AVAILABILITY (L3HA) 17
3.3. EXAMPLE 3: CONFIGURING THE TELEMETRY DISPATCHER 17
3.4. EXAMPLE 4: CONFIGURING RABBITMQ FILE DESCRIPTOR LIMIT 17
3.5. EXAMPLE 5: ENABLING AND DISABLING PARAMETERS 17
3.6. IDENTIFYING PARAMETERS TO MODIFY 18

CHAPTER 4. CONFIGURATION HOOKS 20
4.1. FIRST BOOT: CUSTOMIZING FIRST BOOT CONFIGURATION 20
4.2. PRE-CONFIGURATION: CUSTOMIZING SPECIFIC OVERCLOUD ROLES 21
4.3. PRE-CONFIGURATION: CUSTOMIZING ALL OVERCLOUD ROLES 23
4.4. POST-CONFIGURATION: CUSTOMIZING ALL OVERCLOUD ROLES 25
4.5. PUPPET: CUSTOMIZING HIERADATA FOR ROLES 27
4.6. PUPPET: CUSTOMIZING HIERADATA FOR INDIVIDUAL NODES 28
4.7. PUPPET: APPLYING CUSTOM MANIFESTS 28

CHAPTER 5. OVERCLOUD REGISTRATION 30
5.1. REGISTERING THE OVERCLOUD WITH AN ENVIRONMENT FILE 30
5.2. EXAMPLE 1: REGISTERING TO THE CUSTOMER PORTAL 32
5.3. EXAMPLE 2: REGISTERING TO A RED HAT SATELLITE 6 SERVER 33
5.4. EXAMPLE 3: REGISTERING TO A RED HAT SATELLITE 5 SERVER 33
5.5. EXAMPLE 4: REGISTERING THROUGH A HTTP PROXY 34
5.6. ADVANCED REGISTRATION METHODS 34

CHAPTER 6. COMPOSABLE SERVICES AND CUSTOM ROLES 36
6.1. SUPPORTED CUSTOM ROLE ARCHITECTURE 36
6.2. GUIDELINES AND LIMITATIONS 36
6.3. ROLES 37
6.3.1. Examining the roles_data File 37
6.3.2. Creating a role_data File 38
6.3.3. Examining Role Parameters 39
6.3.4. Creating a New Role 41
6.4. COMPOSABLE SERVICES 42
6.4.1. Examining Composable Service Architecture 43
6.4.2. Adding and Removing Services from Roles 44
6.4.3. Enabling Disabled Services 45
6.4.4. Creating a Generic Node with No Services 45
6.5. ARCHITECTURES 46
6.5.1. Service Architecture: Monolithic Controller 46
6.5.2. Service Architecture: Split Controller 48
6.5.3. Service Architecture: Standalone Roles 51

6.6. COMPOSABLE SERVICE REFERENCE 65

CHAPTER 7. CONTAINERIZED SERVICES 75
7.1. CONTAINERIZED SERVICE ARCHITECTURE 75
7.2. CONTAINERIZED SERVICE PARAMETERS 75
7.3. MODIFYING OPENSTACK PLATFORM CONTAINERS 76

CHAPTER 8. ISOLATING NETWORKS 79
8.1. CREATING CUSTOM INTERFACE TEMPLATES 80
8.2. CREATING A NETWORK ENVIRONMENT FILE 87
8.3. ASSIGNING OPENSTACK SERVICES TO ISOLATED NETWORKS 89
8.4. SELECTING NETWORKS TO DEPLOY 90

CHAPTER 9. USING COMPOSABLE NETWORKS 96
9.1. DEFINING A COMPOSABLE NETWORK 96
9.1.1. Define Network Interface Configuration for Composable Networks 96
9.1.2. Assign Composable Networks to Services 97
9.1.3. Define the Routed Networks 98
9.2. NETWORKING WITH ROUTED SPINE-LEAF 99
9.3. HARDWARE PROVISIONING WITH ROUTED SPINE-LEAF 100
9.3.1. Example VLAN Provisioning Network 100
9.3.2. Example VXLAN Provisioning Network 101
9.3.3. Network Topology for Provisioning 102
9.3.4. Topology Diagram 103
9.3.5. Assign IP Addresses to the Custom Roles 104
9.3.6. Assign Routes for the Roles 104
9.3.7. Custom NIC definitions 107

CHAPTER 10. CONTROLLING NODE PLACEMENT 124
10.1. ASSIGNING SPECIFIC NODE IDS 124
10.2. ASSIGNING CUSTOM HOSTNAMES 125
10.3. ASSIGNING PREDICTABLE IPS 125
10.4. ASSIGNING PREDICTABLE VIRTUAL IPS 127

CHAPTER 11. ENABLING SSL/TLS ON OVERCLOUD PUBLIC ENDPOINTS 129
11.1. INITIALIZING THE SIGNING HOST 129
11.2. CREATING A CERTIFICATE AUTHORITY 129
11.3. ADDING THE CERTIFICATE AUTHORITY TO CLIENTS 129
11.4. CREATING AN SSL/TLS KEY 130
11.5. CREATING AN SSL/TLS CERTIFICATE SIGNING REQUEST 130
11.6. CREATING THE SSL/TLS CERTIFICATE 131
11.7. ENABLING SSL/TLS 131
11.8. INJECTING A ROOT CERTIFICATE 132
11.9. CONFIGURING DNS ENDPOINTS 133
11.10. ADDING ENVIRONMENT FILES DURING OVERCLOUD CREATION 134
11.11. UPDATING SSL/TLS CERTIFICATES 134

CHAPTER 12. ENABLING SSL/TLS ON INTERNAL AND PUBLIC ENDPOINTS WITH IDENTITY MANAGEMENT 135
12.1. ADD THE UNDERCLOUD TO THE CA 135
12.2. ADD THE UNDERCLOUD TO IDM 135
12.3. CONFIGURE OVERCLOUD DNS 136
12.4. CONFIGURE OVERCLOUD TO USE NOVAJOIN 136

CHAPTER 13. DEBUG MODES 139

CHAPTER 14. POLICIES **140**

CHAPTER 15. STORAGE CONFIGURATION **141**
15.1. CONFIGURING NFS STORAGE 141
15.2. CONFIGURING CEPH STORAGE 142
15.3. USING AN EXTERNAL OBJECT STORAGE CLUSTER 142
15.4. CONFIGURING THIRD PARTY STORAGE 143

CHAPTER 16. SECURITY ENHANCEMENTS **145**
16.1. MANAGING THE OVERCLOUD FIREWALL 145
16.2. CHANGING THE SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) STRINGS 146
16.3. CHANGING THE SSL/TLS CIPHER AND RULES FOR HAPROXY 147
16.4. USING THE OPEN VSWITCH FIREWALL 148
16.5. USING SECURE ROOT USER ACCESS 148

CHAPTER 17. FENCING THE CONTROLLER NODES **150**
17.1. REVIEW THE PREREQUISITES 150
17.2. ENABLE FENCING 150
17.3. TEST FENCING 151

CHAPTER 18. CONFIGURING MONITORING TOOLS **153**

CHAPTER 19. CONFIGURING NETWORK PLUGINS **154**
19.1. FUJITSU CONVERGED FABRIC (C-FABRIC) 154
19.2. FUJITSU FOS SWITCH 154

CHAPTER 20. CONFIGURING IDENTITY **156**
20.1. REGION NAME 156

CHAPTER 21. OTHER CONFIGURATIONS **157**
21.1. CONFIGURING EXTERNAL LOAD BALANCING 157
21.2. CONFIGURING IPV6 NETWORKING 157

APPENDIX A. NETWORK ENVIRONMENT OPTIONS **158**

APPENDIX B. NETWORK INTERFACE TEMPLATE EXAMPLES **161**
B.1. CONFIGURING INTERFACES 161
B.2. CONFIGURING ROUTES AND DEFAULT ROUTES 161
B.3. CONFIGURE INTERFACE MAPPING 162
B.4. USING THE NATIVE VLAN FOR FLOATING IPS 163
B.5. USING THE NATIVE VLAN ON A TRUNKED INTERFACE 163
B.6. CONFIGURING JUMBO FRAMES 164

CHAPTER 22. NETWORK INTERFACE PARAMETERS **166**
22.1. INTERFACE OPTIONS 166
22.2. VLAN OPTIONS 166
22.3. OVS BOND OPTIONS 167
22.4. OVS BRIDGE OPTIONS 168
22.5. LINUX BOND OPTIONS 169
22.6. LINUX BRIDGE OPTIONS 170

APPENDIX C. OPEN VSWITCH BONDING OPTIONS **172**
C.1. CHOOSING A BOND MODE 172
C.2. BONDING OPTIONS 173

CHAPTER 1. INTRODUCTION

The Red Hat OpenStack Platform director provides a set of tools to provision and create a fully featured OpenStack environment, also known as the Overcloud. The Director Installation and Usage Guide covers the preparation and configuration of the Overcloud. However, a proper production-level Overcloud might require additional configuration, including:

- Basic network configuration to integrate the Overcloud into your existing network infrastructure.
- Network traffic isolation on separate VLANs for certain OpenStack network traffic types.
- SSL configuration to secure communication on public endpoints
- Storage options such as NFS, iSCSI, Red Hat Ceph Storage, and multiple third-party storage devices.
- Registration of nodes to the Red Hat Content Delivery Network or your internal Red Hat Satellite 5 or 6 server.
- Various system level options.
- Various OpenStack service options.

This guide provides instructions for augmenting your Overcloud through the director. At this point, the director has registered the nodes and configured the necessary services for Overcloud creation. Now you can customize your Overcloud using the methods in this guide.

NOTE

The examples in this guide are optional steps for configuring the Overcloud. These steps are only required to provide the Overcloud with additional functionality. Use only the steps that apply to the needs of your environment.

CHAPTER 2. UNDERSTANDING HEAT TEMPLATES

The custom configurations in this guide use Heat templates and environment files to define certain aspects of the Overcloud. This chapter provides a basic introduction to Heat templates so that you can understand the structure and format of these templates in the context of the Red Hat OpenStack Platform director.

2.1. HEAT TEMPLATES

The director uses Heat Orchestration Templates (HOT) as a template format for its Overcloud deployment plan. Templates in HOT format are mostly expressed in YAML format. The purpose of a template is to define and create a *stack*, which is a collection of resources that heat creates, and the configuration of the resources. Resources are objects in OpenStack and can include compute resources, network configuration, security groups, scaling rules, and custom resources.

The structure of a Heat template has three main sections:

Parameters

These are settings passed to heat, which provides a way to customize a stack, and any default values for parameters without passed values. These are defined in the **`parameters`** section of a template.

Resources

These are the specific objects to create and configure as part of a stack. OpenStack contains a set of core resources that span across all components. These are defined in the **`resources`** section of a template.

Output

These are values passed from heat after the stack’s creation. You can access these values either through the heat API or client tools. These are defined in the **`output`** section of a template.

Here is an example of a basic heat template:

```
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 ] }
```

This template uses the resource type **`type: OS::Nova::Server`** to create an instance called **`my_instance`** with a particular flavor, image, and key. The stack can return the value of **`instance_name`**, which is called **`My Cirros Instance`**.

When Heat processes a template it creates a stack for the template and a set of child stacks for resource templates. This creates a hierarchy of stacks that descend from the main stack you define with your template. You can view the stack hierarchy using this following command:

```
$ heat stack-list --show-nested
```

2.2. ENVIRONMENT FILES

An environment file is a special type of template that provides customization for your Heat templates. This includes three key parts:

Resource Registry

This section defines custom resource names, linked to other heat templates. This essentially provides a method to create custom resources that do not exist within the core resource collection. These are defined in the **`resource_registry`** section of an environment file.

Parameters

These are common settings you apply to the top-level template's parameters. For example, if you have a template that deploys nested stacks, such as resource registry mappings, the parameters only apply to the top-level template and not templates for the nested resources. Parameters are defined in the **`parameters`** section of an environment file.

Parameter Defaults

These parameters modify the default values for parameters in all templates. For example, if you have a Heat template that deploys nested stacks, such as resource registry mappings,the parameter defaults apply to all templates. In other words, the top-level template and those defining all nested resources. The parameter defaults are defined in the **`parameter_defaults`** section of an environment file.

IMPORTANT

It is recommended to use **`parameter_defaults`** instead of **`parameters`** When creating custom environment files for your Overcloud. This is so the parameters apply to all stack templates for the Overcloud.

An example of a basic environment file:

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

```
parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

For example, this environment file (**my_env.yaml**) might be included when creating a stack from a certain Heat template (**my_template.yaml**). The **my_env.yaml** files creates a new resource type called **OS::Nova::Server::MyServer**. The **myserver.yaml** file is a Heat template file that provides an implementation for this resource type that overrides any built-in ones. You can include the **OS::Nova::Server::MyServer** resource in your **my_template.yaml** file.

The **MyIP** applies a parameter only to the main Heat template that deploys along with this environment file. In this example, it only applies to the parameters in **my_template.yaml**.

The **NetworkName** applies to both the main Heat template (in this example, **my_template.yaml**) and the templates associated with resources included the main template, such as the **OS::Nova::Server::MyServer** resource and its **myserver.yaml** template in this example.

2.3. CORE OVERCLOUD HEAT TEMPLATES

The director contains a core heat template collection for the Overcloud. This collection is stored in **/usr/share/openstack-tripleo-heat-templates**.

There are many heat templates and environment files in this collection. However, the main files and directories to note in this template collection are:

overcloud.j2.yaml

This is the main template file used to create the Overcloud environment. This file uses Jinja2 syntax to iterate over certain sections in the template to create custom roles. The Jinja2 formatting is rendered into YAML during the overcloud deployment process.

overcloud-resource-registry-puppet.j2.yaml

This is the main environment file used to create the Overcloud environment. It provides a set of configurations for Puppet modules stored on the Overcloud image. After the director writes the Overcloud image to each node, Heat starts the Puppet configuration for each node using the resources registered in this environment file. This file uses Jinja2 syntax to iterate over certain sections in the template to create custom roles. The Jinja2 formatting is rendered into YAML during the overcloud deployment process.

roles_data.yaml

A file that defines the roles in an overcloud and maps services to each role.

network_data.yaml

A file that defines the networks in an overcloud and their properties such as subnets, allocation pools, and VIP status. The default **network_data** file only contains the default networks: External, Internal Api, Storage, Storage Management, Tenant, and Management. You can create a custom **network_data** file and add it to your**openstack overcloud deploy** command with the **-n** option.

plan-environment.yaml

A file that defines the metadata for your overcloud plan. This includes the plan name, main template to use, and environment files to apply to the overcloud.

capabilities-map.yaml

A mapping of environment files for an overcloud plan. Use this file to describe and enable environment files through the director’s web UI. Custom environment files detected in the **environments** directory in an overcloud plan but not defined in the **capabilities-map.yaml** are listed in the **Other** subtab of **2 Specify Deployment Configuration > Overall Settings** on the web UI.

environments

Contains additional Heat environment files that you can use with your Overcloud creation. These environment files enable extra functions for your resulting OpenStack environment. For example, the directory contains an environment file for enabling Cinder NetApp backend storage (**cinder-netapp-config.yaml**). Any environment files detected in this directory that are not defined in the **capabilities-map.yaml** file are listed in the **Other** subtab of **2 Specify Deployment Configuration > Overall Settings** in the director’s web UI.

network

A set of Heat templates to help create isolated networks and ports.

puppet

Templates mostly driven by configuration with puppet. The aforementioned **overcloud-resource-registry-puppet.j2.yaml** environment file uses the files in this directory to drive the application of the Puppet configuration on each node.

puppet/services

A directory containing heat templates for all services in the composable service architecture.

extraconfig

Templates used to enable extra functionality. For example, the **extraconfig/pre_deploy/rhel-registration** director provides the ability to register your nodes' Red Hat Enterprise Linux operating systems to the Red Hat Content Delivery network or your own Red Hat Satellite server.

firstboot

Provides example **first_boot** scripts that the director uses when initially creating the nodes.

2.4. PLAN ENVIRONMENT METADATA

A plan environment metadata file allows you to define metadata about your overcloud plan. This information is used when importing and exporting your overcloud plan, plus used during the overcloud creation from your plan.

A plan environment metadata file includes the following parameters:

version

The version of the template.

name

The name of the overcloud plan and the container in OpenStack Object Storage (swift) used to store the plan files.

template

The core parent template to use for the overcloud deployment. This is most often **overcloud.yaml**, which is the rendered version of the **overcloud.yaml.j2** template.

environments

Defines a list of environment files to use. Specify the path of each environment file with the **path** sub-parameter.

parameter_defaults

A set of parameters to use in your overcloud. This functions in the same way as the **parameter_defaults** section in a standard environment file.

passwords

A set of parameters to use for overcloud passwords. This functions in the same way as the **parameter_defaults** section in a standard environment file. Normally, the director automatically populates this section with randomly generated passwords.

workflow_parameters

Allows you to provide a set of parameters to OpenStack Workflow (mistral) namespaces. You can use this to calculate and automatically generate certain overcloud parameters.

The following is an example of the syntax of a plan environment file:

```
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
```

You can include the plan environment metadata file with the **openstack overcloud deploy** command using the **-p** option. For example:

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

You can also view plan metadata for an existing overcloud plan using the following command:

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

2.5. CAPABILITIES MAP

The capabilities map provides a mapping of environment files in your plan and their dependencies. Use this file to describe and enable environment files through the director's web UI. Custom environment files detected in an overcloud plan but not listed in the **capabilities-map.yaml** are listed in the **Other** subtab of **2 Specify Deployment Configuration > Overall Settings** on the web UI.

The default file is located at **/usr/share/openstack-tripleo-heat-templates/capabilities-map.yaml**.

The following is an example of the syntax for a capabilities map:

```
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 The **`topics`** parameter contains a list of sections in the UI's deployment configuration. Each topic is displayed as a single screen of environment options and contains multiple environment groups, which you define with the **`environment_groups`** parameter. Each topic can have a plain text **`title`** and **`description`**.

2 The **`environment_groups`** parameter lists groupings of environment files in the UI's deployment configuration. For example, on a storage topic, you might have an environment group for Ceph-related environment files. Each environment group can have a plain text **`title`** and **`description`**.

3 The **`environments`** parameter shows all environment files that belong to an environment group. The **`file`** parameter is the location of the environment file. Each environment entry can have a plain text **`title`** and **`description`**.

4 5 The **`requires`** parameter is a list of dependencies for an environment file. In this example, both **`environment_file_1.yaml`** and **`environment_file_2.yaml`** require you to enable **`dependent_environment_file.yaml`** too.

NOTE

Red Hat OpenStack Platform uses this file to add access to features to the director UI. It is recommended not to modify this file as newer versions of Red Hat OpenStack Platform might override this file.

2.6. INCLUDING ENVIRONMENT FILES IN OVERCLOUD CREATION

The deployment command (**`openstack overcloud deploy`**) uses the **`-e`** option to include an environment file to customize your Overcloud. You can include as many environment files as necessary. However, the order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence. For example, you might have two environment files:

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'
```

Then deploy with both environment files included:

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

In this example, both environment files contain a common resource type (**OS::TripleO::NodeExtraConfigPost**) and a common parameter (**TimeZone**). The **openstack overcloud deploy** command runs through the following process:

1. Loads the default configuration from the core Heat template collection as per the **--template** option.

2. Applies the configuration from **environment-file-1.yaml**, which overrides any common settings from the default configuration.

3. Applies the configuration from **environment-file-2.yaml**, which overrides any common settings from the default configuration and **environment-file-1.yaml**.

This results in the following changes to the default configuration of the Overcloud:

- **OS::TripleO::NodeExtraConfigPost** resource is set to **/home/stack/templates/template-2.yaml** as per **environment-file-2.yaml**.

- **TimeZone** parameter is set to **Hongkong** as per **environment-file-2.yaml**.

- **RabbitFDLimit** parameter is set to **65536** as per **environment-file-1.yaml**. **environment-file-2.yaml** does not change this value.

This provides a method for defining custom configuration to the your Overcloud without values from multiple environment files conflicting.

2.7. USING CUSTOMIZED CORE HEAT TEMPLATES

When creating the overcloud, the director uses a core set of Heat templates located in **`/usr/share/openstack-tripleo-heat-templates`**. If you want to customize this core template collection, use a Git workflow to track changes and merge updates. Use the following git processes to help manage your custom template collection:

Initializing a Custom Template Collection

Use the following procedure to create an initial Git repository containing the Heat template collection:

1. Copy the template collection to the **`stack`** users directory. This example copies the collection to the **`~/templates`** directory:

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

2. Change to the custom template directory and initialize a Git repository:

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

3. Stage all templates for the initial commit:

   ```
   $ git add *
   ```

4. Create an initial commit:

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

This creates an initial **`master`** branch containing the latest core template collection. Use this branch as the basis for your custom branch and merge new template versions to this branch.

Creating a Custom Branch and Committing Changes

Use a custom branch to store your changes to the core template collection. Use the following procedure to create a **`my-customizations`** branch and add customizations to it:

1. Create the **`my-customizations`** branch and switch to it:

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

2. Edit the files in the custom branch.

3. Stage the changes in git:

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

4. Commit the changes to the custom branch:

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

This adds your changes as commits to the **`my-customizations`** branch. When the **`master`** branch updates, you can rebase **`my-customizations`** off **`master`**, which causes git to add these commits on to the updated template collection. This helps track your customizations and replay them on future template updates.

Updating the Custom Template Collection:

When updating the undercloud, the **`openstack-tripleo-heat-templates`** package might also update. When this occurs, use the following procedure to update your custom template collection:

1. Save the **`openstack-tripleo-heat-templates`** package version as an environment variable:

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

2. Change to your template collection directory and create a new branch for the updated templates:

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

3. Remove all files in the branch and replace them with the new versions:

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

4. Add all templates for the initial commit:

   ```
   $ git add *
   ```

5. Create a commit for the package update:

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

6. Merge the branch into master. If using a Git management system (such as GitLab) use the management workflow. If using git locally, merge by switching to the **`master`** branch and run the **`git merge`** command:

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

The **`master`** branch now contains the latest version of the core template collection. You can now rebase the **`my-customization`** branch from this updated collection.

Rebasing the Custom Branch

Use the following procedure to update the **`my-customization`** branch,:

1. Change to the **`my-customizations`** branch:

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

2. Rebase the branch off **`master`**:

   ```
   $ git rebase master
   ```

This updates the **`my-customizations`** branch and replays the custom commits made to this branch.

If git reports any conflicts during the rebase, use this procedure:

1. Check which files contain the conflicts:

   ```
   $ git status
   ```

2. Resolve the conflicts of the template files identified.

3. Add the resolved files

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

4. Continue the rebase:

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

Deploying Custom Templates

Use the following procedure to deploy the custom template collection:

1. Make sure you have switched to the **`my-customization`** branch:

   ```
   git checkout my-customizations
   ```

2. Run the **`openstack overcloud deploy`** command with the **`--templates`** option to specify your local template directory:

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

NOTE

The director uses the default template directory (**`/usr/share/openstack-tripleo-heat-templates`**) if you specify the **`--templates`** option without a directory.

IMPORTANT

Red Hat recommends using the methods in Chapter 4, *Configuration Hooks* instead of modifying the heat template collection.

CHAPTER 3. PARAMETERS

Each Heat template in the director's template collection contains a **`parameters`** section. This section defines all parameters specific to a particular overcloud service. This includes the following:

- **`overcloud.j2.yaml`** - Default base parameters
- **`roles_data.yaml`** - Default parameters for composable roles
- **`puppet/services/*.yaml`** - Default parameters for specific services

You can modify the values for these parameters using the following method:

1. Create an environment file for your custom parameters.
2. Include your custom parameters in the **`parameter_defaults`** section of the environment file.
3. Include the environment file with the **`openstack overcloud deploy`** command.

The next few sections contain examples to demonstrate how to configure specific parameters for services in the **`puppet/services`** directory.

3.1. EXAMPLE 1: CONFIGURING THE TIMEZONE

The Heat template for setting the timezone (**`puppet/services/time/timezone.yaml`**) contains a **`TimeZone`** parameter. If you leave the **`TimeZone`** parameter blank, the overcloud sets the time to **`UTC`** as a default. The director recognizes the standard timezone names defined in the timezone database **`/usr/share/zoneinfo/`**. For example, if you wanted to set your time zone to **`Japan`**, you would examine the contents of **`/usr/share/zoneinfo`** to locate a suitable entry:

```
$ ls /usr/share/zoneinfo/
Africa      Asia       Canada   Cuba   EST      GB       GMT-0      HST
iso3166.tab  Kwajalein  MST      NZ-CHAT   posix       right      Turkey
UTC       Zulu
America     Atlantic   CET      EET    EST5EDT  GB-Eire  GMT+0
Iceland  Israel       Libya      MST7MDT  Pacific   posixrules  ROC
UCT        WET
Antarctica  Australia  Chile    Egypt  Etc      GMT      Greenwich
Indian   Jamaica      MET        Navajo   Poland    PRC         ROK
Universal  W-SU
Arctic      Brazil     CST6CDT  Eire   Europe   GMT0     Hongkong   Iran
Japan        Mexico     NZ       Portugal  PST8PDT     Singapore  US
zone.tab
```

The output listed above includes time zone files, and directories containing additional time zone files. For example, **`Japan`** is an individual time zone file in this result, but **`Africa`** is a directory containing additional time zone files:

```
$ ls /usr/share/zoneinfo/Africa/
Abidjan      Algiers  Bamako  Bissau       Bujumbura   Ceuta
Dar_es_Salaam  El_Aaiun  Harare        Kampala   Kinshasa    Lome
Lusaka  Maseru     Monrovia  Niamey       Porto-Novo  Tripoli
Accra        Asmara   Bangui  Blantyre     Cairo       Conakry  Djibouti
Freetown  Johannesburg  Khartoum  Lagos       Luanda      Malabo  Mbabane
```

```
Nairobi   Nouakchott   Sao_Tome    Tunis
Addis_Ababa  Asmera   Banjul  Brazzaville  Casablanca  Dakar    Douala
Gaborone  Juba          Kigali    Libreville  Lubumbashi  Maputo
Mogadishu  Ndjamena  Ouagadougou  Timbuktu    Windhoek
```

Add the entry in an environment file to set your timezone to **`Japan`**:

```
parameter_defaults:
  TimeZone: 'Japan'
```

3.2. EXAMPLE 2: DISABLING LAYER 3 HIGH AVAILABILITY (L3HA)

The Heat template for the OpenStack Networking (neutron) API (**`puppet/services/neutron-api.yaml`**) contains a parameter to enable and disable Layer 3 High Availability (L3HA). The default for the parameter is **`false`**. However, you can enable it using the following in an environment file:

```
parameter_defaults:
  NeutronL3HA: true
```

3.3. EXAMPLE 3: CONFIGURING THE TELEMETRY DISPATCHER

The OpenStack Telemetry (**`ceilometer`**) service includes a component for a time series data storage (**`gnocchi`**). The **`puppet/services/ceilometer-base.yaml`** Heat Template allows you to switch between **`gnocchi`** and the standard database. You accomplish this with the **`CeilometerMeterDispatcher`** parameter, which you set to either:

- **`gnocchi`** - Use the new time series database for Ceilometer dispatcher. This is the default option.
- **`database`** - Use the standard database for the Ceilometer dispatcher.

To switch to a standard database, add the following to an environment file:

```
parameter_defaults:
  CeilometerMeterDispatcher: database
```

3.4. EXAMPLE 4: CONFIGURING RABBITMQ FILE DESCRIPTOR LIMIT

For certain configurations, you might need to increase the file descriptor limit for the RabbitMQ server. The **`puppet/services/rabbitmq.yaml`** Heat template allows you to set the **`RabbitFDLimit`** parameter to the limit you require. Add the following to an environment file.

```
parameter_defaults:
  RabbitFDLimit: 65536
```

3.5. EXAMPLE 5: ENABLING AND DISABLING PARAMETERS

In some case, you might need to initially set a parameters during a deployment, then disable the parameter for a future deployment operation, such as updates or scaling operations. For example, to include a custom RPM during the overcloud creation, you would include the following:

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

If you need to disable this parameter from a future deployment, it is not enough to remove the parameter. Instead, you set the parameter to an empty value:

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

This ensures the parameter is no longer set for subsequent deployments operations.

3.6. IDENTIFYING PARAMETERS TO MODIFY

Red Hat OpenStack Platform director provides many parameters for configuration. In some cases, you might experience difficulty identifying a certain option to configure and the corresponding director parameter. If there is an option you want to configure through the director, use the following workflow to identify and map the option to a specific overcloud parameter:

1. Identify the option you aim to configure. Make a note of the service that uses the option.

2. Check the corresponding Puppet module for this option. The Puppet modules for Red Hat OpenStack Platform are located under **/etc/puppet/modules** on the director node. Each module corresponds to a particular service. For example, the **keystone** module corresponds to the OpenStack Identity (keystone).

    - If the Puppet module contains a variable that controls the chosen option, move to the next step.

    - If the Puppet module does not contain a variable that controls the chosen option, then no hieradata exists for this option. If possible, you can set the option manually after the overcloud completes deployment.

3. Check the director's core Heat template collection for the Puppet variable in the form of hieradata. The templates in **puppet/services/*** usually correspond to the Puppet modules of the same services. For example, the **puppet/services/keystone.yaml** template provides hieradata to the **keystone** module.

    - If the Heat template sets hieradata for the Puppet variable, the template should also disclose the director-based parameter to modify.

    - If the Heat template does not set hieradata for the Puppet variable, use the configuration hooks to pass the hieradata using an environment file. See Section 4.5, "Puppet: Customizing Hieradata for Roles" for more information on customizing hieradata.

Workflow Example

You might aim to change the notification format for OpenStack Identity (keystone). Using the workflow, you would:

1. Identify the OpenStack parameter to configure (**notification_format**).

2. Search the **keystone** Puppet module for the **notification_format** setting. For example:

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

    In this case, the **keystone** module manages this option using the

keystone::notification_format variable.

3. Search the **keystone** service template for this variable. For example:

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

   The output shows the director using the **KeystoneNotificationFormat** parameter to set the **keystone::notification_format** hieradata.

The following table shows the eventual mapping:

Director Parameter	Puppet Hieradata	OpenStack Identity (keystone) option
KeystoneNotificationFormat	**keystone::notification_format**	**notification_format**

This means setting the **KeystoneNotificationFormat** in an overcloud’s environment file would set the **notification_format** option in the **keystone.conf** file during the overcloud’s configuration.

CHAPTER 4. CONFIGURATION HOOKS

The configuration hooks provide a method to inject your own configuration functions into the Overcloud deployment process. This includes hooks for injecting custom configuration before and after the main Overcloud services configuration and hook for modifying and including Puppet-based configuration.

4.1. FIRST BOOT: CUSTOMIZING FIRST BOOT CONFIGURATION

The director provides a mechanism to perform configuration on all nodes upon the initial creation of the Overcloud. The director achieves this through **`cloud-init`**, which you can call using the **`OS::TripleO::NodeUserData`** resource type.

In this example, you will update the nameserver with a custom IP address on all nodes. You must first create a basic heat template (**`/home/stack/templates/nameserver.yaml`**) that runs a script to append each node's **`resolv.conf`** with a specific nameserver. You can use the **`OS::TripleO::MultipartMime`** resource type to send the configuration script.

```
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}
```

Next, create an environment file (**`/home/stack/templates/firstboot.yaml`**) that registers your heat template as the **`OS::TripleO::NodeUserData`** resource type.

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

To add the first boot configuration, add the environment file to the stack along with your other environment files when first creating the Overcloud. For example:

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

The **-e** applies the environment file to the Overcloud stack.

This adds the configuration to all nodes when they are first created and boot for the first time. Subsequent inclusions of these templates, such as updating the Overcloud stack, does not run these scripts.

IMPORTANT

You can only register the **OS::TripleO::NodeUserData** to one heat template. Subsequent usage overrides the heat template to use.

4.2. PRE-CONFIGURATION: CUSTOMIZING SPECIFIC OVERCLOUD ROLES

IMPORTANT

Previous versions of this document used the **OS::TripleO::Tasks::*PreConfig** resources to provide pre-configuration hooks on a per role basis. The director's Heat template collection requires dedicated use of these hooks, which means you should not use them for custom use. Instead, use the **OS::TripleO::*ExtraConfigPre** hooks outlined below.

The Overcloud uses Puppet for the core configuration of OpenStack components. The director provides a set of hooks to provide custom configuration for specific node roles after the first boot completes and before the core configuration begins. These hooks include:

OS::TripleO::ControllerExtraConfigPre

Additional configuration applied to Controller nodes before the core Puppet configuration.

OS::TripleO::ComputeExtraConfigPre

Additional configuration applied to Compute nodes before the core Puppet configuration.

OS::TripleO::CephStorageExtraConfigPre

Additional configuration applied to Ceph Storage nodes before the core Puppet configuration.

OS::TripleO::ObjectStorageExtraConfigPre

Additional configuration applied to Object Storage nodes before the core Puppet configuration.

OS::TripleO::BlockStorageExtraConfigPre

Additional configuration applied to Block Storage nodes before the core Puppet configuration.

OS::TripleO::[ROLE]ExtraConfigPre

Additional configuration applied to custom nodes before the core Puppet configuration. Replace **[ROLE]** with the composable role name.

In this example, you first create a basic heat template (**/home/stack/templates/nameserver.yaml**) that runs a script to write to a node's**resolv.conf** with a variable nameserver.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

  parameters:
    server:
      type: json
    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]}
```

In this example, the **resources** section contains the following:

CustomExtraConfigPre

This defines a software configuration. In this example, we define a Bash **script** and Heat replaces **_NAMESERVER_IP_** with the value stored in the **nameserver_ip** parameter.

CustomExtraDeploymentPre

This executes a software configuration, which is the software configuration from the **CustomExtraConfigPre** resource. Note the following:

- The **config** parameter makes a reference to the**CustomExtraConfigPre** resource so Heat knows what configuration to apply.

- The **server** parameter retrieves a map of the Overcloud nodes. This parameter is provided by the parent template and is mandatory in templates for this hook.

- The **actions** parameter defines when to apply the configuration. In this case, we only apply the configuration when the Overcloud is created. Possible actions include **CREATE**, **UPDATE**, **DELETE**, **SUSPEND**, and **RESUME**.

- **input_values** contains a parameter called **deploy_identifier**, which stores the

DeployIdentifier from the parent template. This parameter provides a timestamp to the resource for each deployment update. This ensures the resource reapplies on subsequent overcloud updates.

Next, create an environment file (**/home/stack/templates/pre_config.yaml**) that registers your heat template to the role-based resource type. For example, to apply only to Controller nodes, use the **ControllerExtraConfigPre** hook:

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

To apply the configuration, add the environment file to the stack along with your other environment files when creating or updating the Overcloud. For example:

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

This applies the configuration to all Controller nodes before the core configuration begins on either the initial Overcloud creation or subsequent updates.

IMPORTANT

You can only register each resource to only one Heat template per hook. Subsequent usage overrides the Heat template to use.

4.3. PRE-CONFIGURATION: CUSTOMIZING ALL OVERCLOUD ROLES

The Overcloud uses Puppet for the core configuration of OpenStack components. The director provides a hook to configure all node types after the first boot completes and before the core configuration begins:

OS::TripleO::NodeExtraConfig

Additional configuration applied to all nodes roles before the core Puppet configuration.

In this example, you first create a basic heat template (**/home/stack/templates/nameserver.yaml**) that runs a script to append each node’s **resolv.conf** with a variable nameserver.

```
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]}
```

In this example, the **resources** section contains the following:

CustomExtraConfigPre

This defines a software configuration. In this example, we define a Bash **script** and Heat replaces **_NAMESERVER_IP_** with the value stored in the **nameserver_ip** parameter.

CustomExtraDeploymentPre

This executes a software configuration, which is the software configuration from the **CustomExtraConfigPre** resource. Note the following:

- The **config** parameter makes a reference to the**CustomExtraConfigPre** resource so Heat knows what configuration to apply.
- The **server** parameter retrieves a map of the Overcloud nodes. This parameter is provided by the parent template and is mandatory in templates for this hook.
- The **actions** parameter defines when to apply the configuration. In this case, we only apply the configuration when the Overcloud is created. Possible actions include **CREATE**, **UPDATE**, **DELETE**, **SUSPEND**, and **RESUME**.
- The **input_values** parameter contains a sub-parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update. This ensures the resource reapplies on subsequent overcloud updates.

Next, create an environment file (**/home/stack/templates/pre_config.yaml**) that registers your heat template as the **OS::TripleO::NodeExtraConfig** resource type.

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

To apply the configuration, add the environment file to the stack along with your other environment files when creating or updating the Overcloud. For example:

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

This applies the configuration to all nodes before the core configuration begins on either the initial Overcloud creation or subsequent updates.

IMPORTANT

You can only register the **OS::TripleO::NodeExtraConfig** to only one Heat template. Subsequent usage overrides the Heat template to use.

4.4. POST-CONFIGURATION: CUSTOMIZING ALL OVERCLOUD ROLES

IMPORTANT

Previous versions of this document used the **OS::TripleO::Tasks::*PostConfig** resources to provide post-configuration hooks on a per role basis. The director's Heat template collection requires dedicated use of these hooks, which means you should not use them for custom use. Instead, use the **OS::TripleO::NodeExtraConfigPost** hook outlined below.

A situation might occur where you have completed the creation of your Overcloud but want to add additional configuration to all roles, either on initial creation or on a subsequent update of the Overcloud. In this case, you use the following post-configuration hook:

OS::TripleO::NodeExtraConfigPost

Additional configuration applied to all nodes roles after the core Puppet configuration.

In this example, you first create a basic heat template (**/home/stack/templates/nameserver.yaml**) that runs a script to append each node's **resolv.conf** with a variable nameserver.

```
heat_template_version: 2014-10-16

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}
```

In this example, the **resources** section contains the following:

CustomExtraConfig

This defines a software configuration. In this example, we define a Bash **script** and Heat replaces **_NAMESERVER_IP_** with the value stored in the **nameserver_ip** parameter.

CustomExtraDeployments

This executes a software configuration, which is the software configuration from the **CustomExtraConfig** resource. Note the following:

- The **config** parameter makes a reference to the**CustomExtraConfig** resource so Heat knows what configuration to apply.
- The **servers** parameter retrieves a map of the Overcloud nodes. This parameter is provided by the parent template and is mandatory in templates for this hook.
- The **actions** parameter defines when to apply the configuration. In this case, we only apply the configuration when the Overcloud is created. Possible actions include **CREATE**, **UPDATE**, **DELETE**, **SUSPEND**, and **RESUME**.
- **input_values** contains a parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update. This ensures the resource reapplies on subsequent overcloud updates.

Next, create an environment file (**/home/stack/templates/post_config.yaml**) that registers your heat template as the **OS::TripleO::NodeExtraConfigPost:** resource type.

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

To apply the configuration, add the environment file to the stack along with your other environment files when creating or updating the Overcloud. For example:

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

This applies the configuration to all nodes after the core configuration completes on either initial Overcloud creation or subsequent updates.

IMPORTANT

You can only register the **`OS::TripleO::NodeExtraConfigPost`** to only one Heat template. Subsequent usage overrides the Heat template to use.

4.5. PUPPET: CUSTOMIZING HIERADATA FOR ROLES

The Heat template collection contains a set of parameters to pass extra configuration to certain node types. These parameters save the configuration as hieradata for the node’s Puppet configuration. These parameters are:

ControllerExtraConfig

Configuration to add to all Controller nodes.

ComputeExtraConfig

Configuration to add to all Compute nodes.

BlockStorageExtraConfig

Configuration to add to all Block Storage nodes.

ObjectStorageExtraConfig

Configuration to add to all Object Storage nodes

CephStorageExtraConfig

Configuration to add to all Ceph Storage nodes

[ROLE]ExtraConfig

Configuration to add to a composable role. Replace **`[ROLE]`** with the composable role name.

ExtraConfig

Configuration to add to all nodes.

To add extra configuration to the post-deployment configuration process, create an environment file that contains these parameters in the **`parameter_defaults`** section. For example, to increase the reserved memory for Compute hosts to 1024 MB and set the VNC keymap to Japanese:

```
parameter_defaults:
  ComputeExtraConfig:
```

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

Include this environment file when running **openstack overcloud deploy**.

IMPORTANT

You can only define each parameter once. Subsequent usage overrides previous values.

4.6. PUPPET: CUSTOMIZING HIERADATA FOR INDIVIDUAL NODES

You can set Puppet hieradata for individual nodes using the Heat template collection. To accomplish this, you need to acquire the system UUID saved as part of the introspection data for a node:

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

This outputs a system UUID. For example:

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

Use this system UUID in an environment file that defines node-specific hieradata and registers the **per_node.yaml** template to a pre-configuration hook. For example:

```
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" ]}}'
```

Include this environment file when running **openstack overcloud deploy**.

The **per_node.yaml** template generates a set of heiradata files on nodes that correspond to each system UUID and contains the hieradata you defined. If a UUID is not defined, the resulting hieradata file is empty. In the previous example, the **per_node.yaml** template runs on all Compute nodes (as per the **OS::TripleO::ComputeExtraConfigPre** hook), but only the Compute node with system UUID **F5055C6C-477F-47FB-AFE5-95C6928C407F** receives hieradata.

This provides a method of tailoring each node to specific requirements.

4.7. PUPPET: APPLYING CUSTOM MANIFESTS

In certain circumstances, you might need to install and configure some additional components to your Overcloud nodes. You can achieve this with a custom Puppet manifest that applies to nodes on after the main configuration completes. As a basic example, you might intend to install **motd** to each node. The process for accomplishing is to first create a Heat template (**/home/stack/templates/custom_puppet_config.yaml**) that launches Puppet configuration.

```
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}
```

This includes the **`/home/stack/templates/motd.pp`** within the template and passes it to nodes for configuration. The **`motd.pp`** file itself contains the Puppet classes to install and configure **`motd`**.

Next, create an environment file (**`/home/stack/templates/puppet_post_config.yaml`**) that registers your heat template as the **`OS::TripleO::NodeExtraConfigPost:`** resource type.

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

And finally include this environment file along with your other environment files when creating or updating the Overcloud stack:

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

This applies the configuration from **`motd.pp`** to all nodes in the Overcloud.

CHAPTER 5. OVERCLOUD REGISTRATION

The Overcloud provides a method to register nodes to either the Red Hat Content Delivery Network, a Red Hat Satellite 5 server, or a Red Hat Satellite 6 server.

5.1. REGISTERING THE OVERCLOUD WITH AN ENVIRONMENT FILE

Copy the registration files from the Heat template collection:

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

Edit the **`~/templates/rhel-registration/environment-rhel-registration.yaml`** and modify the following values to suit your registration method and details.

General Parameters

rhel_reg_method

Choose the registration method. Either **`portal`**, **`satellite`**, or **`disable`**.

rhel_reg_type

The type of unit to register. Leave blank to register as a **`system`**

rhel_reg_auto_attach

Automatically attach compatible subscriptions to this system. Set to **`true`** to enable. To disable this feature, remove this parameter from your environment file.

rhel_reg_service_level

The service level to use for auto attachment.

rhel_reg_release

Use this parameter to set a release version for auto attachment. Leave blank to use the default from Red Hat Subscription Manager.

rhel_reg_pool_id

The subscription pool ID to use. Use this if not auto-attaching subscriptions. To locate this ID, run **`sudo subscription-manager list --available --all --matches="*OpenStack*"`** from the undercloud node, and use the resulting **`Pool ID`** value.

rhel_reg_sat_url

The base URL of the Satellite server to register Overcloud nodes. Use the Satellite's HTTP URL and not the HTTPS URL for this parameter. For example, use http://satellite.example.com and not https://satellite.example.com. The Overcloud creation process uses this URL to determine whether the server is a Red Hat Satellite 5 or Red Hat Satellite 6 server. If a Red Hat Satellite 6 server, the Overcloud obtains the **`katello-ca-consumer-latest.noarch.rpm`** file, registers with **`subscription-manager`**, and installs **`katello-agent`**. If a Red Hat Satellite 5 server, the Overcloud obtains the **`RHN-ORG-TRUSTED-SSL-CERT`** file and registers with **`rhnreg_ks`**.

rhel_reg_server_url

The hostname of the subscription service to use. The default is for Customer Portal Subscription Management, subscription.rhn.redhat.com. If this option is not used, the system is registered with Customer Portal Subscription Management. The subscription server URL uses the form of https://hostname:port/prefix.

rhel_reg_base_url

Gives the hostname of the content delivery server to use to receive updates. The default is https://cdn.redhat.com. Since Satellite 6 hosts its own content, the URL must be used for systems registered with Satellite 6. The base URL for content uses the form of https://hostname:port/prefix.

rhel_reg_org

The organization to use for registration. To locate this ID, run `sudo subscription-manager orgs` from the undercloud node. Enter your Red Hat credentials when prompted, and use the resulting `Key` value.

rhel_reg_environment

The environment to use within the chosen organization.

rhel_reg_repos

A comma-separated list of repositories to enable.

rhel_reg_activation_key

The activation key to use for registration.

rhel_reg_user; rhel_reg_password

The username and password for registration. If possible, use activation keys for registration.

rhel_reg_machine_name

The machine name. Leave this as blank to use the hostname of the node.

rhel_reg_force

Set to `true` to force your registration options. For example, when re-registering nodes.

rhel_reg_sat_repo

The repository containing Red Hat Satellite 6's management tools, such as `katello-agent`. Check the correct repository name corresponds to your Red Hat Satellite version and check that the repository is synchronized on the Satellite server. For example, `rhel-7-server-satellite-tools-6.2-rpms` corresponds to Red Hat Satellite 6.2.

Upgrade Parameters

UpdateOnRHELRegistration

If set to `True`, this triggers an update of the overcloud packages after registration completes. Set to `False` by default.

HTTP Proxy Parameters

rhel_reg_http_proxy_host

The hostname for the HTTP proxy. For example: `proxy.example.com`.

rhel_reg_http_proxy_port

The port for HTTP proxy communication. For example: `8080`.

rhel_reg_http_proxy_username

The username to access the HTTP proxy.

rhel_reg_http_proxy_password

The password to access the HTTP proxy.

IMPORTANT

If using a proxy server, ensure all overcloud nodes have a route to the host defined in the **`rhel_reg_http_proxy_host`** parameter. Without a route to this host, **`subscription-manager`** will time out and cause deployment failure.

The deployment command (**`openstack overcloud deploy`**) uses the **`-e`** option to add environment files. Add both **`~/templates/rhel-registration/environment-rhel-registration.yaml`** and **`~/templates/rhel-registration/rhel-registration-resource-registry.yaml`**. For example:

```
$ 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
```

IMPORTANT

Registration is set as the **`OS::TripleO::NodeExtraConfig`** Heat resource. This means you can only use this resource for registration. See Section 4.2, “Pre-Configuration: Customizing Specific Overcloud Roles” for more information.

5.2. EXAMPLE 1: REGISTERING TO THE CUSTOMER PORTAL

The following registers the overcloud nodes to the Red Hat Customer Portal using the **`my-openstack`** activation key and subscribes to pool **`1a85f9223e3d5e43013e3d6e8ff506fd`**.

```
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-12-rpms,rhel-7-server-rhceph-2-osd-rpms,rhel-7-server-rhceph-2-
mon-rpms,rhel-7-server-rhceph-2-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. EXAMPLE 2: REGISTERING TO A RED HAT SATELLITE 6 SERVER

The following registers the overcloud nodes to a Red Hat Satellite 6 Server at sat6.example.com and uses the **`my-openstack`** activation key to subscribe to pool **`1a85f9223e3d5e43013e3d6e8ff506fd`**. In this situation, the activation key also provides the repositories to enable.

```
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. EXAMPLE 3: REGISTERING TO A RED HAT SATELLITE 5 SERVER

The following registers the overcloud nodes to a Red Hat Satellite 5 Server at sat5.example.com, uses the **`my-openstack`** activation key, and automatically attaches subscriptions. In this situation, the activation key also provides the repositories to enable.

```
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. EXAMPLE 4: REGISTERING THROUGH A HTTP PROXY

The following sample parameters set the HTTP proxy settings for your desired registration method:

```
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. ADVANCED REGISTRATION METHODS

In some situations, you might aim to register different roles to different subscription types. For example, you might aim to only subscribe Controller nodes to an OpenStack Platform subscription and Ceph Storage nodes to a Ceph Storage subscription. This section provides some advanced registration methods to help with assigning separate subscriptions to different roles.

Configuration Hooks

One method is to write role-specific scripts and include them with a role-specific hook. For example, the following snippet could be added to the **`OS::TripleO::ControllerExtraConfigPre`** resource's template, which ensures only the Controller nodes receive these subscription details.

```
ControllerRegistrationConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: script
    config:
      str_replace:
        template: |
          #!/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-12-rpms \
            --enable rhel-7-server-rhceph-2-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}
```

The script uses a set of **subscription-manager** commands to register the system, attach the subscription, and enable the required repositories.

For more information about hooks, see Chapter 4, *Configuration Hooks*.

Ansible-Based Configuration

You can perform Ansible-based registration on specific roles using the director's dynamic inventory script. For example, you might aim to register Controller nodes using the following play:

```
---
- 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-12-rpms
      - rhel-7-server-rhceph-2-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 }}"
```

This play contains three tasks: - Register the node using an activation key - Disable any auto-enabled repositories - Enable only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

After deploying the overcloud, you can run the following command so that Ansible executes the playbook (**ansible-osp-registration.yml**) against your overcloud:

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

This command does the following: - Runs the dynamic inventory script to get a list of host and their groups - Applies the playbook tasks to the nodes in the group defined in the playbook's **hosts** parameter, which in this case is the **Controller** group.

For more information on the running Ansible automation on your overcloud, see "Running Ansible Automation" in the *Director Installation and Usage* guide.

CHAPTER 6. COMPOSABLE SERVICES AND CUSTOM ROLES

The Overcloud usually consists of nodes in predefined roles such as Controller nodes, Compute nodes, and different storage node types. Each of these default roles contains a set of services defined in the core Heat template collection on the director node. However, the architecture of the core Heat templates provide methods to:

- Create custom roles
- Add and remove services from each role

This allows the possibility to create different combinations of services on different roles. This chapter explores the architecture of custom roles, composable services, and methods for using them.

6.1. SUPPORTED CUSTOM ROLE ARCHITECTURE

Only a limited number of composable service combinations have been tested and verified. Red Hat supports the following architectures when using custom roles and composable services:

Architecture 1 - Monolithic Controller

All controller services are contained within one Controller role. This is the default. See Section 6.5.1, “Service Architecture: Monolithic Controller” for more details.

Architecture 2 - Split Controller

The controller services are split into two roles:

- Controller PCMK - Core Pacemaker-managed services such as database and load balancing
- Controller Systemd - 'systemd`-managed OpenStack Platform services

See Section 6.5.2, “Service Architecture: Split Controller” for more details.

Architecture 3 - Standalone roles

Use Architecture 1 or Architecture 2, except split the OpenStack Platform services into custom roles. See Section 6.5.3, “Service Architecture: Standalone Roles” for more details.

6.2. GUIDELINES AND LIMITATIONS

Note the following guidelines and limitations for the composable node architecture.

For **`systemd`** services:

- You can assign **`systemd`** managed services to supported standalone custom roles.
- You can create additional custom roles after the initial deployment and deploy them to scale existing **`systemd`** services.

For Pacemaker-managed services:

- You can assign Pacemaker managed services to supported standalone custom roles.
- Pacemaker has a 16 node limit. If you assign the Pacemaker service (**`OS::TripleO::Services::Pacemaker`**) to 16 nodes, any subsequent nodes must use the Pacemaker Remote service (**`OS::TripleO::Services::PacemakerRemote`**) instead. You cannot have the Pacemaker service and Pacemaker Remote service on the same role.

- Do not include the Pacemaker service (**`OS::TripleO::Services::Pacemaker`**) on roles that do not contain Pacemaker managed services.

- You cannot scale up or scale down a custom roles that contains **`OS::TripleO::Services::Pacemaker`** or **`OS::TripleO::Services::PacemakerRemote`** services.

General Limitations:

- You cannot change custom roles and composable services during the upgrade process from Red Hat OpenStack Platform 11 to 12.

- You cannot modify the list of services for any role after deploying an Overcloud. Modifying the service lists after Overcloud deployment can cause deployment errors and leave orphaned services on nodes.

6.3. ROLES

6.3.1. Examining the roles_data File

The Overcloud creation process defines its roles using a **`roles_data`** file. The **`roles_data`** file contains a YAML-formatted list of the roles. The following is a shortened example of the **`roles_data`** syntax:

```
- 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
    ...
```

The core Heat template collection contains a default **`roles_data`** file located at **`/usr/share/openstack-tripleo-heat-templates/roles_data.yaml`**. The default file defines the following role types:

- **`Controller`**

- **`Compute`**

- **`BlockStorage`**

- **`ObjectStorage`**

- **CephStorage**.

The **openstack overcloud deploy** command includes this file during deployment. You can override this file with a custom **roles_data** file using the **-r** argument. For example:

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

6.3.2. Creating a role_data File

Although you can manually create a custom **roles_data** file, you can also automatically generating the file using individual role templates. The director provides a several commands to manage role templates and automatically generate a custom **roles_data** file.

To list the default role templates, use the **openstack overcloud role list** command:

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

To see the role’s YAML definition, use the **openstack overcloud role show** command:

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

To generate a custom **roles_data** file, use the **openstack overcloud roles generate** command to join multiple predefined roles into a single file. For example, the following command joins the **Controller**, **Compute**, and **Networker** roles into a single file:

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

The **-o** defines the name of the file to create.

This creates a custom **roles_data** file. However, the previous example uses the **Controller** and **Networker** roles, which both contain the same networking agents. This means the networking services scale from **Controller** to the **Networker** role. The overcloud balance the load for networking services between the **Controller** and **Networker** nodes.

To make this **Networker** role standalone, you can create your own custom **Controller** role, as well as any other role needed. This allows you to easily generate a **roles_data** file from your own custom roles.

Copy the directory from the core Heat template collection to the **stack** user’s home directory:

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

Add or modify the custom role files in this directory. Use the **--roles-path** option with any of the aforementioned role sub-commands to use this directory as the source for your custom roles. For example:

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

This generates a single **my_roles_data.yaml** file from the individual roles in the **~/roles** directory.

NOTE

The default roles collection also contains the **ControllerOpenStack** role, which does not include services for **Networker**, **Messaging**, and **Database** roles. You can use the **ControllerOpenStack** combined with with the standalone **Networker**, **Messaging**, and **Database** roles.

6.3.3. Examining Role Parameters

Each role uses the following parameters:

name

(Mandatory) The name of the role, which is a plain text name with no spaces or special characters. Check that the chosen name does not cause conflicts with other resources. For example, use **Networker** as a name instead of **Network**. For recommendations on role names, see Section 6.5.2, “Service Architecture: Split Controller” for examples.

description

(Optional) A plain text description for the role.

tags

(Optional) A YAML list of tags that o define role properties. Use this parameter to define the primary role with both the **controller** and **primary** tags together:

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

IMPORTANT

If you do not tag the primary role, the first role defined becomes the primary role. Ensure this role is the Controller role.

networks

A list of networks to configure on the role. Default networks include **External**, **InternalApi**, **Storage**, **StorageMgmt**, **Tenant**, and **Management**.

CountDefault

(Optional) Defines the default number of nodes to deploy for this role.

HostnameFormatDefault

(Optional) Defines the default hostname format for the role. The default naming convention uses the following format:

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

For example, the default Controller nodes are named:

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

disable_constraints

(Optional) Defines whether to disable OpenStack Compute (nova) and OpenStack Image Storage (glance) constraints when deploying with the director. Used when deploying an overcloud with pre-provisioned nodes. For more information, see "Configuring a Basic Overcloud using Pre-Provisioned Nodes" in the *Director Installation and Usage Guide*.

disable_upgrade_deployment

(Optional) Defines whether to disable upgrades for a specific role. This provides a method to upgrade individual nodes in a role and ensure availability of services. For example, the Compute and Swift Storage roles use this parameter.

upgrade_batch_size

(Optional) Defines the number of tasks to execute in a batch during the upgrade. A task counts as one upgrade step per node. The default batch size is 1, which means the upgrade process executes a single upgrade step on each node one at a time. Increasing the batch size increases the number of tasks executed simultaneously on nodes

ServicesDefault

(Optional) Defines the default list of services to include on the node. SeeSection 6.4.1, “Examining Composable Service Architecture” for more information.

These parameters provide a means to create new roles and also define which services to include.

The **openstack overcloud deploy** command integrates the parameters from the **roles_data** file into some of the Jinja2-based templates. For example, at certain points, the **overcloud.j2.yaml** Heat template iterates over the list of roles from **roles_data.yaml** and creates parameters and resources specific to each respective role.

The resource definition for each role in the **overcloud.j2.yaml** Heat template appears as the following snippet:

```
  {{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]}
...
```

This snippet shows how the Jinja2-based template incorporates the **{{role.name}}** variable to define the name of each role as a **OS::Heat::ResourceGroup** resource. This in turn uses each **name** parameter from the **roles_data** file to name each respective **OS::Heat::ResourceGroup** resource.

6.3.4. Creating a New Role

In this example, the aim is to create a new **Horizon** role to host the OpenStack Dashboard (**horizon**) only. In this situation, you create a custom **roles** directory that includes the new role information.

Create a custom copy of the default **roles** directory:

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

Create a new file called **~/roles/Horizon.yaml** and create a new **Horizon** role containing base and core OpenStack Dashboard services. For example:

```
- 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::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

It is also a good idea to set the **CountDefault** to **1** so that a default Overcloud always includes the **Horizon** node.

If scaling the services in an existing overcloud, keep the existing services on the **Controller** role. If creating a new overcloud and you want the OpenStack Dashboard to remain on the standalone role, remove the OpenStack Dashboard components from the **Controller** role definition:

```
- 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
    ...
```

Generate the new **roles_data** file using the **roles** directory as the source:

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

You might need to define a new flavor for this role so that you can tag specific nodes. For this example, use the following commands to create a **horizon** flavor:

```
$ 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
```

Tag nodes into the new flavor using the following command:

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

Define the Horizon node count and flavor using the following environment file snippet:

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

Include the new **roles_data** file and environment file when running the **openstack overcloud deploy** command. For example:

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

When the deployment completes, this creates a three-node Overcloud consisting of one Controller node, one Compute node, and one Networker node. To view the Overcloud’s list of nodes, run the following command:

```
$ openstack server list
```

6.4. COMPOSABLE SERVICES

6.4.1. Examining Composable Service Architecture

The core Heat template collection contains a collection of composable service templates in the **puppet/services** subdirectory. You can view these services with the following command:

```
$ ls /usr/share/openstack-tripleo-heat-templates/puppet/services
```

Each service template contains a description that identifies its purpose. For example, the **keystone.yaml** service template contains the following description:

```
description: >
 OpenStack Identity (`keystone`) service configured with Puppet
```

These service templates are registered as resources specific to a Red Hat OpenStack Platform deployment. This means you can call each resource using a unique Heat resource namespace defined in the **overcloud-resource-registry-puppet.j2.yaml** file. All services use the **OS::TripleO::Services** namespace for their resource type. For example, the **keystone.yaml** service template is registered to the **OS::TripleO::Services::Keystone** resource type:

```
grep "OS::TripleO::Services::Keystone" /usr/share/openstack-tripleo-heat-
templates/overcloud-resource-registry-puppet.j2.yaml
  OS::TripleO::Services::Keystone: puppet/services/keystone.yaml
```

The **overcloud.j2.yaml** Heat template includes a section of Jinja2-based code to define a service list for each custom role in the **roles_data.yaml** file:

```
{{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([])}}
```

For the default roles, this creates the following service list parameters: **ControllerServices**, **ComputeServices**, **BlockStorageServices**, **ObjectStorageServices**, and **CephStorageServices**.

You define the default services for each custom role in the **roles_data.yaml** file. For example, the default Controller role contains the following content:

```
- 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
...
```

These services are then defined as the default list for the **ControllerServices** parameter.

You can also use an environment file to override the default list for the service parameters. For example, you can define **ControllerServices** as a **parameter_default** in an environment file to override the services list from the **roles_data.yaml** file.

6.4.2. Adding and Removing Services from Roles

The basic method of adding or removing services involves creating a copy of the default service list for a node role and then adding or removing services. For example, you might aim to remove OpenStack Orchestration (**heat**) from the Controller nodes. In this situation, create a custom copy of the default **roles** directory:

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

Edit the **~/roles/Controller.yaml** file and modify the service list for the**ServicesDefault** parameter. Scroll to the OpenStack Orchestration services and remove them:

```
    - 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
```

Generate the new **roles_data** file. For example:

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

Include this new **roles_data** file when running the **openstack overcloud deploy** command. For example:

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

This deploys an Overcloud without OpenStack Orchestration services installed on the Controller nodes.

NOTE

You can also disable services in the **roles_data** file using a custom environment file. Redirect the services to disable to the **OS::Heat::None** resource. For example:

```
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
```

6.4.3. Enabling Disabled Services

Some services are disabled by default. These services are registered as null operations (**OS::Heat::None**) in the **overcloud-resource-registry-puppet.j2.yaml** file. For example, the Block Storage backup service (**cinder-backup**) is disabled:

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

To enable this service, include an environment file that links the resource to its respective Heat templates in the **puppet/services** directory. Some services have predefined environment files in the **environments** directory. For example, the Block Storage backup service uses the **environments/cinder-backup.yaml** file, which contains the following:

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

This overrides the default null operation resource and enables the service. Include this environment file when running the **openstack overcloud deploy** command.

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

TIP

For another example of how to enable disabled services, see the Installation section of the OpenStack Data Processing guide. This section contains instructions on how to enable the OpenStack Data Processing service (**sahara**) on the overcloud.

6.4.4. Creating a Generic Node with No Services

Red Hat OpenStack Platform provides the ability to create generic Red Hat Enterprise Linux 7 nodes without any OpenStack services configured. This is useful in situations where you need to host software outside of the core Red Hat OpenStack Platform environment. For example, OpenStack Platform provides integration with monitoring tools such as Kibana and Sensu (see Monitoring Tools Configuration Guide). While Red Hat does not provide support for the monitoring tools themselves, the director can create a generic Red Hat Enterprise Linux 7 node to host these tools.

NOTE

The generic node still uses the base **overcloud-full** image rather than a base Red Hat Enterprise Linux 7 image. This means the node has some Red Hat OpenStack Platform software installed but not enabled or configured.

Creating a generic node requires a new role without a **ServicesDefault** list:

```
- name: Generic
```

Include the role in your custom **roles_data** file (**roles_data_with_generic.yaml**). Make sure to keep the existing **Controller** and **Compute** roles.

You can also include an environment file (**generic-node-params.yaml**) to specify how many generic Red Hat Enterprise Linux 7 nodes you require and the flavor when selecting nodes to provision. For example:

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

Include both the roles file and the environment file when running the **openstack overcloud deploy** command. For example:

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

This deploys a three-node environment with one Controller node, one Compute node, and one generic Red Hat Enterprise Linux 7 node.

6.5. ARCHITECTURES

6.5.1. Service Architecture: Monolithic Controller

The default architecture for composable services uses a monolithic Controller that contains the core Red Hat OpenStack Platform Services. These default services are defined in the roles file included with the director’s Heat template collection (**/usr/share/openstack-tripleo-heat-templates/roles_data.yaml**).

IMPORTANT

Some services are disabled by default. See Section 6.4.3, “Enabling Disabled Services” for information on how to enable these services.

```
- name: Controller # the 'primary' role goes first
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMds
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRbdMirror
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::CinderBackendDellPs
    - OS::TripleO::Services::CinderBackendDellSc
    - OS::TripleO::Services::CinderBackendNetApp
    - OS::TripleO::Services::CinderBackendScaleIO
    - OS::TripleO::Services::Congress
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::RabbitMQ
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::Redis
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::MongoDb
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaPlacement
    - OS::TripleO::Services::NovaMetadata
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaConsoleauth
    - OS::TripleO::Services::NovaVncProxy
    - OS::TripleO::Services::Ec2Api
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::CeilometerApi
    - OS::TripleO::Services::CeilometerCollector
    - OS::TripleO::Services::CeilometerExpirer
    - OS::TripleO::Services::CeilometerAgentCentral
    - OS::TripleO::Services::CeilometerAgentNotification
    - OS::TripleO::Services::Horizon
    - OS::TripleO::Services::GnocchiApi
    - OS::TripleO::Services::GnocchiMetricd
```

```
      - OS::TripleO::Services::GnocchiStatsd
      - OS::TripleO::Services::ManilaApi
      - OS::TripleO::Services::ManilaScheduler
      - OS::TripleO::Services::ManilaBackendGeneric
      - OS::TripleO::Services::ManilaBackendNetapp
      - OS::TripleO::Services::ManilaBackendCephFs
      - OS::TripleO::Services::ManilaShare
      - OS::TripleO::Services::AodhApi
      - OS::TripleO::Services::AodhEvaluator
      - OS::TripleO::Services::AodhNotifier
      - OS::TripleO::Services::AodhListener
      - OS::TripleO::Services::SaharaApi
      - OS::TripleO::Services::SaharaEngine
      - OS::TripleO::Services::IronicApi
      - OS::TripleO::Services::IronicConductor
      - OS::TripleO::Services::NovaIronic
      - OS::TripleO::Services::TripleoPackages
      - OS::TripleO::Services::TripleoFirewall
      - OS::TripleO::Services::OpenDaylightApi
      - OS::TripleO::Services::OpenDaylightOvs
      - OS::TripleO::Services::SensuClient
      - OS::TripleO::Services::FluentdClient
      - OS::TripleO::Services::Collectd
      - OS::TripleO::Services::BarbicanApi
      - OS::TripleO::Services::PankoApi
      - OS::TripleO::Services::Tacker
      - OS::TripleO::Services::Zaqar
      - OS::TripleO::Services::OVNDBs
      - OS::TripleO::Services::NeutronML2FujitsuCfab
      - OS::TripleO::Services::NeutronML2FujitsuFossw
      - OS::TripleO::Services::CinderHPELeftHandISCSI
      - OS::TripleO::Services::Etcd
      - OS::TripleO::Services::AuditD
      - OS::TripleO::Services::OctaviaApi
      - OS::TripleO::Services::OctaviaHealthManager
      - OS::TripleO::Services::OctaviaHousekeeping
      - OS::TripleO::Services::OctaviaWorker
```

6.5.2. Service Architecture: Split Controller

You can split the services on the Controller nodes into two separate roles:

- **Controller PCMK** - Contains only the core services that Pacemaker manages including database and load balancing

- **Controller systemd** - Contains all OpenStack services

The remaining default roles (Compute, Ceph Storage, Object Storage, Block Storage) remain unaffected.

Use the following tables as a guide to creating a split controller architecture.

IMPORTANT

Some services are disabled by default. See Section 6.4.3, “Enabling Disabled Services” for information on how to enable these services.

Controller PCMK

The following services are the minimum services required for the Controller PCMK role.

```
- name: Controller
  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::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::ManilaBackendGeneric
    - OS::TripleO::Services::ManilaBackendNetapp
    - OS::TripleO::Services::ManilaBackendCephFs
    - OS::TripleO::Services::ManilaShare
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::RabbitMQ
    - OS::TripleO::Services::Redis
```

Controller systemd

The following table represents the services available on the Controller systemd role:

```
- name: ControllerSystemd
  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::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::AodhApi
    - OS::TripleO::Services::AodhEvaluator
    - OS::TripleO::Services::AodhListener
    - OS::TripleO::Services::AodhNotifier
    - OS::TripleO::Services::CeilometerAgentCentral
    - OS::TripleO::Services::CeilometerAgentNotification
    - OS::TripleO::Services::CeilometerApi
    - OS::TripleO::Services::CeilometerCollector
    - OS::TripleO::Services::CeilometerExpirer
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GnocchiApi
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    - OS::TripleO::Services::Horizon
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::ManilaApi
    - OS::TripleO::Services::ManilaScheduler
    - OS::TripleO::Services::MongoDb
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronCorePluginML2OVN
    - OS::TripleO::Services::NeutronCorePluginMidonet
    - OS::TripleO::Services::NeutronCorePluginNuage
    - OS::TripleO::Services::NeutronCorePluginOpencontrail
    - OS::TripleO::Services::NeutronCorePluginPlumgrid
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::NovaConsoleauth
    - OS::TripleO::Services::NovaIronic
    - OS::TripleO::Services::NovaPlacement
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaVncProxy
    - OS::TripleO::Services::OpenDaylightApi
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::PankoApi
```

```
      - OS::TripleO::Services::SaharaApi
      - OS::TripleO::Services::SaharaEngine
      - OS::TripleO::Services::SwiftProxy
      - OS::TripleO::Services::SwiftRingBuilder
```

6.5.3. Service Architecture: Standalone Roles

The following tables list the supported custom role collection you can create and scale with the composable service architecture in Red Hat OpenStack Platform. Group these collections together as individual roles and use them to isolate and split services in combination with the previous architectures:

- Section 6.5.1, “Service Architecture: Monolithic Controller”
- Section 6.5.2, “Service Architecture: Split Controller”

IMPORTANT

Some services are disabled by default. See Section 6.4.3, “Enabling Disabled Services” for information on how to enable these services.

Note that all roles use a set of *common services*, which include:

- **OS::TripleO::Services::AuditD**
- **OS::TripleO::Services::CACerts**
- **OS::TripleO::Services::CertmongerUser**
- **OS::TripleO::Services::Collectd**
- **OS::TripleO::Services::ContainersLogrotateCrond**
- **OS::TripleO::Services::Docker**
- **OS::TripleO::Services::FluentdClient**
- **OS::TripleO::Services::Kernel**
- **OS::TripleO::Services::Ntp**
- **OS::TripleO::Services::SensuClient**
- **OS::TripleO::Services::Snmp**
- **OS::TripleO::Services::Timezone**
- **OS::TripleO::Services::TripleoFirewall**
- **OS::TripleO::Services::TripleoPackages**
- **OS::TripleO::Services::Tuned**

Once you have chosen the roles to include in your overcloud, remove the associated services (except for the *common services*) from the main Controller roles. For example, if creating a standalone Keystone role, remove the **OS::TripleO::Services::Apache** and **OS::TripleO::Services::Keystone** services from the Controller nodes. The only exceptions are the services with limited custom role support (see Table 6.1, “Custom Roles Support”).

Click on a role in the following table to view the services associated with it.

Table 6.1. Custom Roles Support

Role	Support Status
Ceph Storage Monitor	Supported
Ceph Storage OSD	Supported
Ceph Storage RadosGW	Limited. If spliting, this service needs to be part of a **Controller systemd** role.
Cinder API	Supported
Controller PCMK	Supported
Database	Supported
Glance	Supported
Heat	Supported
Horizon	Supported
Ironic	Supported
Keystone	Supported
Load Balancer	Supported
Manila	Limited. If spliting, this service needs to be part of a **Controller systemd** role.
Message Bus	Supported
Networker	Supported
Neutron API	Supported
Nova	Supported
Nova Compute	Supported
OpenDaylight	Technical Preview

Role	Support Status
Redis	Supported
Sahara	Limited. If spliting, this service needs to be part of a **Controller systemd** role.
Swift API	Supported
Swift Storage	Supported
Telemetry	Supported

Ceph Storage Monitor

The following services configure Ceph Storage Monitor.

```
- name: CephMon
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephMon
```

Ceph Storage OSD

The following services configure Ceph Storage OSDs.

```
- name: CephStorage
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephOSD
```

Ceph Storage RadosGW

The following services configure Ceph Storage RadosGW. If separating these services, they need to be part of a **Controller systemd** role.

```
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephRgw
```

Cinder API

The following services configure the OpenStack Block Storage API.

```
- name: CinderApi
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderScheduler
```

Controller PCMK

The following services are the minimum services required for the Controller PCMK as a standalone role.

```
- name: Controller
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::ManilaBackendGeneric
    - OS::TripleO::Services::ManilaBackendNetapp
    - OS::TripleO::Services::ManilaBackendCephFs
    - OS::TripleO::Services::ManilaShare
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::Pacemaker
```

This is the same as the Controller PCMK role in the Split Controller Architecture. The difference is you can split the following highly available services to standalone roles:

- Database
- Load Balancer
- Message Bus
- Redis

If not, creating standalone roles for these services, merge the services from these roles into the Controller PCMK standalone role.

Database

The following services configure the main database. The database is MariaDB managed as a Galera cluster using Pacemaker.

```
- name: Database
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::MySQL
```

Glance

The following services configure the OpenStack Image service.

```
- name: Glance
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::GlanceApi
```

Heat

The following services configure the OpenStack Orchestration service.

```
- name: Heat
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
```

Horizon

The following services configure the OpenStack Dashboard.

```
- name: Horizon
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

Ironic

The following services configure the OpenStack Bare Metal Provisioning service.

```
- name: Ironic
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::IronicPxe
```

Note the following:

- Requires access to the Storage network.
- The **`OS::TripleO::Services::IronicApi`** service can exist on either the **`Ironic`** role or the **`Controller`** role depending on your requirements.
- Requires the **`OS::TripleO::Services::NovaIronic`** service on the **`Controller`** role.

Keystone

The following services configure the OpenStack Identity service. When performing minor updates, make sure to update this role before updating other services.

```
- name: Keystone
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Keystone
```

Load Balancer

The following services configure the overcloud’s load balancer. The load balancer is HAProxy managed with Pacemaker.

```
- name: LoadBalancer
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::HAproxy
```

Manila

The following services configure the OpenStack Shared File Systems service. If separating these services, they need to be part of a **Controller systemd** role.

```
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::ManilaApi
    - OS::TripleO::Services::ManilaScheduler
```

Message Bus

The following services configure the messaging queue. The messaging queue is RabbitMQ managed with Pacemaker.

```
- name: MessageBus
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::RabbitMQ
```

Networker

The following services configure the OpenStack Networking agents.

```
- name: Networker
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronOvsAgent
```

Neutron API

The following services configure the OpenStack Networking API.

```
- name: NeutronApi
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronCorePluginML2OVN
    - OS::TripleO::Services::NeutronCorePluginMidonet
    - OS::TripleO::Services::NeutronCorePluginNuage
    - OS::TripleO::Services::NeutronCorePluginOpencontrail
    - OS::TripleO::Services::NeutronCorePluginPlumgrid
```

Nova

The following services configure the OpenStack Compute services.

```
- name: Nova
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::NovaConsoleauth
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaPlacement
    - OS::TripleO::Services::NovaVncProxy
```

Nova Compute

The following services configure an OpenStack Compute node.

```
- name: Compute
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NeutronSriovAgent
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::OpenDaylightOvs
```

OpenDaylight

The following services configure OpenDayLight. **These services are technology preview for Red Hat OpenStack Platform 11**.

```
- name: Opendaylight
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::OpenDaylightApi
    - OS::TripleO::Services::OpenDaylightOvs
```

Redis

The following services configure Redis managed with Pacemaker.

```
- name: Redis
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::Redis
```

Sahara

The following services configure the OpenStack Clustering service. If separating these services, they need to be part of a **Controller systemd** role.

```
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::SaharaApi
    - OS::TripleO::Services::SaharaEngine
```

Swift API

The following services configure the OpenStack Object Storage API.

```
- name: SwiftApi
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftRingBuilder
```

Swift Storage

The following services configure the OpenStack Object Storage service.

```
- name: ObjectStorage
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::SwiftStorage
```

Telemetry

The following services configure the OpenStack Telemetry services.

```
- name: Telemetry
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::AodhApi
    - OS::TripleO::Services::AodhEvaluator
    - OS::TripleO::Services::AodhListener
    - OS::TripleO::Services::AodhNotifier
    - OS::TripleO::Services::CeilometerAgentCentral
    - OS::TripleO::Services::CeilometerAgentNotification
    - OS::TripleO::Services::CeilometerApi
    - OS::TripleO::Services::CeilometerCollector
    - OS::TripleO::Services::CeilometerExpirer
    - OS::TripleO::Services::GnocchiApi
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::MongoDb
    - OS::TripleO::Services::PankoApi
```

6.6. COMPOSABLE SERVICE REFERENCE

The following tables contain all composable service available for Red Hat OpenStack Platform 12:

- Table 6.2, “Services Retained from Previous Versions”
- Table 6.3, “New Services for Red Hat OpenStack Platform 12”

IMPORTANT

Some services are disabled by default. See Section 6.4.3, “Enabling Disabled Services” for information on how to enable these services.

Table 6.2. Services Retained from Previous Versions

Service	Description

Service	Description
OS::TripleO::Services::AodhApi	OpenStack Telemetry Alarming (**aodh**) API service configured with Puppet
OS::TripleO::Services::AodhEvaluator	OpenStack Telemetry Alarming (**aodh**) Evaluator service configured with Puppet
OS::TripleO::Services::AodhListener	OpenStack Telemetry Alarming (**aodh**) Listener service configured with Puppet
OS::TripleO::Services::AodhNotifier	OpenStack Telemetry Alarming (**aodh**) Notifier service configured with Puppet
OS::TripleO::Services::Apache	Apache service configured with Puppet. Note this is typically included automatically with other services which run through Apache.
OS::TripleO::Services::CACerts	HAProxy service configured with Puppet
OS::TripleO::Services::CeilometerAgentCentral	OpenStack Telemetry (**ceilometer**) Central Agent service configured with Puppet
OS::TripleO::Services::CeilometerAgentNotification	OpenStack Telemetry (**ceilometer**) Notification Agent service configured with Puppet
OS::TripleO::Services::CeilometerApi	OpenStack Telemetry (**ceilometer**) API service configured with Puppet
OS::TripleO::Services::CeilometerCollector	OpenStack Telemetry (**ceilometer**) Collector service configured with Puppet
OS::TripleO::Services::CeilometerExpirer	OpenStack Telemetry (**ceilometer**) Expirer service configured with Puppet
OS::TripleO::Services::CephClient	(**Disabled by default**) Ceph Client service
OS::TripleO::Services::CephExternal	(**Disabled by default**) Ceph External service
OS::TripleO::Services::CephMon	(**Disabled by default**) Ceph Monitor service
OS::TripleO::Services::CephOSD	(**Disabled by default**) Ceph OSD service
OS::TripleO::Services::CinderApi	OpenStack Block Storage (**cinder**) API service configured with Puppet
OS::TripleO::Services::CinderBackup	(**Disabled by default**) OpenStack Block Storage (**cinder**) Backup service configured with Puppet

Service	Description
OS::TripleO::Services::CinderScheduler	OpenStack Block Storage (**cinder**) Scheduler service configured with Puppet
OS::TripleO::Services::CinderVolume	OpenStack Block Storage (**cinder**) Volume service (Pacemaker-managed) configured with Puppet
OS::TripleO::Services::ComputeCeilometerAgent	OpenStack Telemetry (**ceilometer**) Compute Agent service configured with Puppet
OS::TripleO::Services::ComputeNeutronCorePlugin	OpenStack Networking (**neutron**) ML2 Plugin configured with Puppet
OS::TripleO::Services::ComputeNeutronL3Agent	(**Disabled by default**) OpenStack Networking (**neutron**) L3 agent for DVR enabled Compute nodes configured with Puppet
OS::TripleO::Services::ComputeNeutronMetadataAgent	(**Disabled by default**) OpenStack Networking (**neutron**) Metadata agent configured with Puppet
OS::TripleO::Services::ComputeNeutronOvsAgent	OpenStack Networking (**neutron**) OVS agent configured with Puppet
OS::TripleO::Services::FluentdClient	(**Disabled by default**) Fluentd client configured with Puppet
OS::TripleO::Services::GlanceApi	OpenStack Image (**glance**) API service configured with Puppet
OS::TripleO::Services::GnocchiApi	OpenStack Telemetry Metrics (**gnocchi**) service configured with Puppet
OS::TripleO::Services::GnocchiMetricd	OpenStack Telemetry Metrics (**gnocchi**) service configured with Puppet
OS::TripleO::Services::GnocchiStatsd	OpenStack Telemetry Metrics (**gnocchi**) service configured with Puppet
OS::TripleO::Services::HAproxy	HAProxy service (Pacemaker-managed) configured with Puppet
OS::TripleO::Services::HeatApi	OpenStack Orchestration (**heat**) API service configured with Puppet
OS::TripleO::Services::HeatApiCfn	OpenStack Orchestration (**heat**) CloudFormation API service configured with Puppet

Service	Description
OS::TripleO::Services::HeatApiCloudwatch	OpenStack Orchestration (**heat**) CloudWatch API service configured with Puppet
OS::TripleO::Services::HeatEngine	OpenStack Orchestration (**heat**) Engine service configured with Puppet
OS::TripleO::Services::Horizon	OpenStack Dashboard (**horizon**) service configured with Puppet
OS::TripleO::Services::IronicApi	(**Disabled by default**) OpenStack Bare Metal Provisioning (**ironic**) API configured with Puppet
OS::TripleO::Services::IronicConductor	(**Disabled by default**) OpenStack Bare Metal Provisioning (**ironic**) conductor configured with Puppet
OS::TripleO::Services::Keepalived	Keepalived service configured with Puppet
OS::TripleO::Services::Kernel	Load kernel modules with kmod and configure kernel options with sysctl
OS::TripleO::Services::Keystone	OpenStack Identity (**keystone**) service configured with Puppet
OS::TripleO::Services::ManilaApi	(**Disabled by default**) OpenStack Shared File Systems (**manila**) API service configured with Puppet
OS::TripleO::Services::ManilaScheduler	(**Disabled by default**) OpenStack Shared File Systems (**manila**) Scheduler service configured with Puppet
OS::TripleO::Services::ManilaShare	(**Disabled by default**) OpenStack Shared File Systems (**manila**) Share service configured with Puppet
OS::TripleO::Services::Memcached	Memcached service configured with Puppet
OS::TripleO::Services::MongoDb	MongoDB service deployment using puppet
OS::TripleO::Services::MySQL	MySQL (Pacemaker-managed) service deployment using puppet
OS::TripleO::Services::NeutronApi	OpenStack Networking (**neutron**) Server configured with Puppet

Service	Description
`OS::TripleO::Services::NeutronCorePlugin`	OpenStack Networking (**`neutron`**) ML2 Plugin configured with Puppet
`OS::TripleO::Services::NeutronCorePluginML2OVN`	OpenStack Networking (**`neutron`**) ML2/OVN plugin configured with Puppet
`OS::TripleO::Services::NeutronCorePluginMidonet`	OpenStack Networking (**`neutron`**) Midonet plugin and services
`OS::TripleO::Services::NeutronCorePluginNuage`	OpenStack Networking (**`neutron`**) Nuage plugin
`OS::TripleO::Services::NeutronCorePluginOpencontrail`	OpenStack Networking (**`neutron`**) Opencontrail plugin
`OS::TripleO::Services::NeutronCorePluginPlumgrid`	OpenStack Networking (**`neutron`**) Plumgrid plugin
`OS::TripleO::Services::NeutronDhcpAgent`	OpenStack Networking (**`neutron`**) DHCP agent configured with Puppet
`OS::TripleO::Services::NeutronL3Agent`	OpenStack Networking (**`neutron`**) L3 agent configured with Puppet
`OS::TripleO::Services::NeutronMetadataAgent`	OpenStack Networking (**`neutron`**) Metadata agent configured with Puppet
`OS::TripleO::Services::NeutronOvsAgent`	OpenStack Networking (**`neutron`**) OVS agent configured with Puppet
`OS::TripleO::Services::NeutronServer`	OpenStack Networking (**`neutron`**) Server configured with Puppet
`OS::TripleO::Services::NeutronSriovAgent`	(**Disabled by default**) OpenStack Neutron SR-IOV nic agent configured with Puppet
`OS::TripleO::Services::NovaApi`	OpenStack Compute (**`nova`**) API service configured with Puppet
`OS::TripleO::Services::NovaCompute`	OpenStack Compute (**`nova`**) Compute service configured with Puppet
`OS::TripleO::Services::NovaConductor`	OpenStack Compute (**`nova`**) Conductor service configured with Puppet
`OS::TripleO::Services::NovaConsoleauth`	OpenStack Compute (**`nova`**) Consoleauth service configured with Puppet

Service	Description
`OS::TripleO::Services::NovaIronic`	(**Disabled by default**) OpenStack Compute (`nova`) service configured with Puppet and using Ironic
`OS::TripleO::Services::NovaLibvirt`	Libvirt service configured with Puppet
`OS::TripleO::Services::NovaScheduler`	OpenStack Compute (`nova`) Scheduler service configured with Puppet
`OS::TripleO::Services::NovaVncProxy`	OpenStack Compute (`nova`) Vncproxy service configured with Puppet
`OS::TripleO::Services::Ntp`	NTP service deployment using Puppet.
`OS::TripleO::Services::OpenDaylightApi`	(**Disabled by default**) OpenDaylight SDN controller
`OS::TripleO::Services::OpenDaylightOvs`	(**Disabled by default**) OpenDaylight OVS configuration
`OS::TripleO::Services::Pacemaker`	Pacemaker service configured with Puppet
`OS::TripleO::Services::RabbitMQ`	RabbitMQ service (Pacemaker-managed) configured with Puppet
`OS::TripleO::Services::Redis`	OpenStack Redis service configured with Puppet
`OS::TripleO::Services::SaharaApi`	(**Disabled by default**) OpenStack Clustering (`sahara`) API service configured with Puppet
`OS::TripleO::Services::SaharaEngine`	(**Disabled by default**) OpenStack Clustering (`sahara`) Engine service configured with Puppet
`OS::TripleO::Services::SensuClient`	(**Disabled by default**) Sensu client configured with Puppet
`OS::TripleO::Services::Snmp`	SNMP client configured with Puppet, to facilitate Ceilometer hardware monitoring in the undercloud. This service is required to enable hardware monitoring.
`OS::TripleO::Services::SwiftProxy`	OpenStack Object Storage (`swift`) Proxy service configured with Puppet
`OS::TripleO::Services::SwiftRingBuilder`	OpenStack Object Storage (`swift`) Ringbuilder

Service	Description
`OS::TripleO::Services::SwiftStorage`	OpenStack Object Storage (`swift`) service configured with Puppet
`OS::TripleO::Services::Timezone`	Composable Timezone service
`OS::TripleO::Services::TripleoFirewall`	Firewall settings
`OS::TripleO::Services::TripleoPackages`	Package installation settings

Table 6.3. New Services for Red Hat OpenStack Platform 12

Service	Description
`OS::TripleO::Services::ApacheTLS`	(**Disabled by default**) Apache service with TLS/SSL enabled. This service is enabled when including Certmonger-based TLS/SSL configuration (`environments/enable-internal-tls.yaml`).
`OS::TripleO::Services::AuditD`	(**Disabled by default**) Implements the auditing service. Enabled when including the auditing service environment file (`environments/auditd.yaml`).
`OS::TripleO::Services::CephMds`	(**Disabled by default**) Ceph Metadata Server (MDS). Enabled when including the Ceph MDS environment file (`environments/services/ceph-mds.yaml`).
`OS::TripleO::Services::CephRbdMirror`	(**Disabled by default**) Ceph Storage RBD Mirroring service. Enabled when including the RBD Mirroring environment file (`environments/services/ceph-rbdmirror.yaml`).
`OS::TripleO::Services::CephRgw`	(**Disabled by default**) Ceph Storage Object Gateway (radosgw). Enabled when including the RadosGW environment file (`environments/ceph-radosgw.yaml`), which also disables OpenStack Object Storage (swift) services.
`OS::TripleO::Services::CinderHPELeftHandISCSI`	(**Disabled by default**) Cinder HPE LeftHand iSCSI backend. Enabled when including the LeftHand iSCSI environment file (`environments/cinder-hpelefthand-config.yaml`).

Service	Description
`OS::TripleO::Services::Collectd`	(**Disabled by default**) The statistics collection daemon. Enabled when including the Collectd environment file (`environments/collectd-environment.yaml`).
`OS::TripleO::Services::Congress`	(**Disabled by default**) OpenStack Policy-as-a-Service (Congress). Enabled when including the Congress environment file (`environments/enable_congress.yaml`).
`OS::TripleO::Services::Etcd`	(**Disabled by default**) Etcd key-value storage. Enabled when including the etcd environment file (`environments/services/etcd.yaml`).
`OS::TripleO::Services::HAProxyInternalTLS`	(**Disabled by default**) Internal network for HAProxy service with TLS/SSL enabled. This service is enabled when including Certmonger-based TLS/SSL configuration (`environments/enable-internal-tls.yaml`).
`OS::TripleO::Services::HAProxyPublicTLS`	(**Disabled by default**) External network for HAProxy service with TLS/SSL enabled. This service is enabled when including Certmonger-based TLS/SSL configuration (`environments/services/haproxy-public-tls-certmonger.yaml`)
`OS::TripleO::Services::ManilaBackendCephFs`	(**Disabled by default**) Manila backend for Ceph Storage. Enabled when including the respective backend environment file (`environments/manila-cephfsnative-config.yaml`).
`OS::TripleO::Services::ManilaBackendGeneric`	(**Disabled by default**) Generic Manila backend. Enabled when including the respective backend environment file (`environments/manila-generic-config.yaml`).
`OS::TripleO::Services::ManilaBackendNetapp`	(**Disabled by default**) Manila backend for NetApp. Enabled when including the respective backend environment file (`environments/manila-netapp-config.yaml`).
`OS::TripleO::Services::MistralApi`	(**Disabled by default**) OpenStack Workflow Service (mistral) API. Enabled when including the mistral environment file (`environments/services/mistral.yaml`).

Service	Description
`OS::TripleO::Services::MistralEngine`	(**Disabled by default**) OpenStack Workflow Service (mistral) Engine. Enabled when including the mistral environment file (**`environments/services/mistral.yaml`**).
`OS::TripleO::Services::MistralExecutor`	(**Disabled by default**) OpenStack Workflow Service (mistral) Execution server. Enabled when including the mistral environment file (**`environments/services/mistral.yaml`**).
`OS::TripleO::Services::MySQLClient`	Database client.
`OS::TripleO::Services::MySQLTLS`	(**Disabled by default**) Database service with TLS/SSL enabled. This service is enabled when including Certmonger-based TLS/SSL configuration (**`environments/enable-internal-tls.yaml`**).
`OS::TripleO::Services::NeutronML2FujitsuCfab`	(**Disabled by default**) Fujitsu C-Fabric plugin for OpenStack network (neutron). Enabled when including the C-Fabric environment file (**`environments/neutron-ml2-fujitsu-cfab.yaml`**).
`OS::TripleO::Services::NeutronML2FujitsuFossw`	(**Disabled by default**) Fujitsu fossw plugin for OpenStack network (neutron). Enabled when including the fossw environment file (**`environments/neutron-ml2-fujitsu-fossw.yaml`**).
`OS::TripleO::Services::NovaMetadata`	OpenStack Compute (nova) metadata service.
`OS::TripleO::Services::NovaPlacement`	OpenStack Compute (nova) placement service.
`OS::TripleO::Services::OctaviaApi`	(**Disabled by default**) OpenStack Load Balancing-as-a-Service (octavia) API. Enabled when including the octavia environment file (**`environments/services/octavia.yaml`**).
`OS::TripleO::Services::OctaviaHealthManager`	(**Disabled by default**) OpenStack Load Balancing-as-a-Service (octavia) Health Manager. Enabled when including the octavia environment file (**`environments/services/octavia.yaml`**).
`OS::TripleO::Services::OctaviaHousekeeping`	(**Disabled by default**) OpenStack Load Balancing-as-a-Service (octavia) Housekeeping service. Enabled when including the octavia environment file (**`environments/services/octavia.yaml`**).

Service	Description
`OS::TripleO::Services::OctaviaWorker`	(**Disabled by default**) OpenStack Load Balancing-as-a-Service (octavia) Worker service. Enabled when including the octavia environment file (**`environments/services/octavia.yaml`**).
`OS::TripleO::Services::OVNDBs`	(**Disabled by default**) OVN databases. Enabled when including the OVN extensions (**`environments/neutron-ml2-ovn.yaml`**).
`OS::TripleO::Services::PankoApi`	OpenStack Telemetry Event Storage (panko).
`OS::TripleO::Services::Sshd`	(**Disabled by default**) SSH daemon configuration. Included as a default service.
`OS::TripleO::Services::Tacker`	(**Disabled by default**) OpenStack NFV Orchestration (tacker). Enabled when including the tacker environment file (**`environments/enable_tacker.yaml`**).
`OS::TripleO::Services::TLSProxyBase`	(**Disabled by default**) Base service for configuring TLS/SSL. This service is enabled when including Certmonger-based TLS/SSL configuration (**`environments/enable-internal-tls.yaml`**).
`OS::TripleO::Services::Zaqar`	(**Disabled by default**) OpenStack Messaging (zaqar). Enabled when including the zaqar environment file (**`environments/services/zaqar.yaml`**).

CHAPTER 7. CONTAINERIZED SERVICES

The director installs the core OpenStack Platform services as containers on the overcloud. This section provides some background information on how containerized services work.

7.1. CONTAINERIZED SERVICE ARCHITECTURE

The director installs the core OpenStack Platform services as containers on the overcloud. The templates for the containerized services are located in the **`/usr/share/openstack-tripleo-heat-templates/docker/services/`**. These templates reference their respective composable service templates. For example, the OpenStack Identity (keystone) containerized service template (**`docker/services/keystone.yaml`**) includes the following resource:

```
  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}
```

The **`type`** refers to the respective OpenStack Identity (keystone) composable service and pulls the **`outputs`** data from that template. The containerized service merges this data with its own container-specific data.

All nodes using containerized services must enable the **`OS::TripleO::Services::Docker`** service. When creating a **`roles_data.yaml`** file for your custom roles configuration, include the the **`OS::TripleO::Services::Docker`** service with the base composable services, as the containerized services. For example, the **`Keystone`** role uses the following role definition:

```
- 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::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Keystone
```

7.2. CONTAINERIZED SERVICE PARAMETERS

Each containerized service template contains an **`outputs`** section that defines a data set passed to the director’s OpenStack Orchestration (heat) service. In addition to the standard composable service parameters (see Section 6.3.3, “Examining Role Parameters”), the template contain a set of parameters specific to the container configuration.

`puppet_config`

Data to pass to Puppet when configuring the service. In the initial overcloud deployment steps, the director creates a set of containers used to configure the service before the actual containerized service runs. This parameter includes the following sub-parameters: +

- **`config_volume`** - The mounted docker volume that stores the configuration.
- **`puppet_tags`** - Tags to pass to Puppet during configuration. These tags are used in OpenStack Platform to restrict the Puppet run to a particular service’s configuration resource. For example, the OpenStack Identity (keystone) containerized service uses the **`keystone_config`** tag to ensure all required only the **`keystone_config`** Puppet resource run on the configuration container.
- **`step_config`** - The configuration data passed to Puppet. This is usually inherited from the referenced composable service.
- **`config_image`** - The container image used to configure the service.

`kolla_config`

A set of container-specific data that defines configuration file locations, directory permissions, and the command to run on the container to launch the service.

`docker_config`

Tasks to run on the service’s configuration container. All tasks are grouped into steps to help the director perform a staged deployment. The steps are: +

- **Step 1** - Load balancer configuration
- **Step 2** - Core services (Database, Redis)
- **Step 3** - Initial configuration of OpenStack Platform service
- **Step 4** - General OpenStack Platform services configuration
- **Step 5** - Service activation

`host_prep_tasks`

Preparation tasks for the bare metal node to accommodate the containerized service.

7.3. MODIFYING OPENSTACK PLATFORM CONTAINERS

Red Hat provides a set of pre-built container images through the Red Hat Container Catalog (**`registry.access.redhat.com`**). It is possible to modify these images and add additional layers to them. This is useful for adding RPMs for certified 3rd party drivers to the containers.

NOTE

To ensure continued support for modified OpenStack Platform container images, ensure that the resulting images comply with the "Red Hat Container Support Policy".

This example shows how to customize the latest **openstack-keystone** image. However, these instructions can also apply to other images:

1. Pull the image you aim to modify. For example, for the **openstack-keystone** image:

   ```
   $ sudo docker pull registry.access.redhat.com/rhosp12/openstack-
   keystone:latest
   ```

2. Check the default user on the original image. For example, for the **openstack-keystone** image:

   ```
   $ sudo docker run -it registry.access.redhat.com/rhosp12/openstack-
   keystone:latest whoami
   root
   ```

   NOTE

   The **openstack-keystone** image uses **root** as the default user. Other images use specific users. For example, **openstack-glance-api** uses **glance** for the default user.

3. Create a **Dockerfile** to build an additional layer on an existing container image. The following is an example that pulls the latest OpenStack Identity (keystone) image from the Container Catalog and installs a custom RPM file to the image:

   ```
   FROM registry.access.redhat.com/rhosp12/openstack-keystone
   MAINTAINER Acme
   LABEL name="rhosp12/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. Build and tag the new image. For example, to build with a local **Dockerfile** stored in the **/home/stack/keystone** directory and tag it to your undercloud's local registry:

   ```
   $ docker build /home/stack/keystone -t
   "192.168.24.1:8787/rhosp12/openstack-keystone-acme:rev1"
   ```

5. Push the resulting image to the undercloud's local registry:

   ```
   $ docker push 192.168.24.1:8787/rhosp12/openstack-keystone-acme:rev1
   ```

6. Edit your overcloud container images environment file (usually **overcloud_images.yaml**) and change the appropriate parameter to use the custom container image.

WARNING

The Container Catalog publishes container images with a complete software stack built into it. When the Container Catalog releases a container image with updates and security fixes, your existing custom container will **not** include these updates and will require rebuilding using the new image version from the Catalog.

CHAPTER 8. ISOLATING NETWORKS

The director provides methods to configure isolated Overcloud networks. This means the Overcloud environment separates network traffic types into different networks, which in turn assigns network traffic to specific network interfaces or bonds. After configuring isolated networks, the director configures the OpenStack services to use the isolated networks. If no isolated networks are configured, all services run on the Provisioning network.

This example uses separate networks for all services:

- NIC1 (Provisioning):
  - Provisioning (also known as Control Plane)
- NIC2 (Control Group)
  - Internal API
  - Storage Management
  - External (Public API)
- NIC3 (Data Group)
  - Tenant Network (VXLAN tunneling)
  - Tenant VLANs / Provider VLANs
  - Storage
  - External VLANs (Floating IP/SNAT)
- NIC4 (Management)
  - Management

In this example, each Overcloud node uses two network interfaces in a bond to serve networks in tagged VLANs. The following network assignments apply to this bond:

Table 8.1. Network Subnet and VLAN Assignments

Network Type	Subnet	VLAN	NIC/Group
Internal API	172.16.0.0/24	201	NIC2 (control)
Tenant	172.17.0.0/24	202	NIC3 (data)
Storage	172.18.0.0/24	203	NIC3 (data)
Storage Management	172.19.0.0/24	204	NIC2 (control)
Management	172.20.0.0/24	205	NIC4 (management)

Network Type	Subnet	VLAN	NIC/Group
External / Floating IP	10.1.1.0/24	100	NIC2 (control) NIC3 (data)

8.1. CREATING CUSTOM INTERFACE TEMPLATES

The Overcloud network configuration requires a set of the network interface templates. You customize these templates to configure the node interfaces on a per role basis. These templates are standard Heat templates in YAML format (see Section 2.1, "Heat Templates"). The director contains a set of example templates to get you started:

- **`/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans`** - Directory containing templates for single NIC with VLANs configuration on a per role basis.
- **`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans`** - Directory containing templates for bonded NIC configuration on a per role basis.
- **`/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics`** - Directory containing templates for multiple NIC configuration using one NIC per role.
- **`/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-linux-bridge-vlans`** - Directory containing templates for single NIC with VLANs configuration on a per role basis and using a Linux bridge instead of an Open vSwitch bridge.

> **NOTE**
>
> These examples only contain templates for the default roles. To define the network interface configuration for a custom role, use these templates as a basis.

For this example, use the default multiple NICs example configuration as a basis. Copy the version located at **`/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics`**.

```
$ cp -r /usr/share/openstack-tripleo-heat-
templates/network/config/multiple-nics ~/templates/nic-configs
```

The command will create a local set of Heat templates that define a network interface configuration with multiple NICs for each role. Each template contains the standard **`parameters`**, **`resources`**, and **`output`** sections.

Parameters

The **`parameters`** section contains all network configuration parameters for network interfaces. This includes information such as subnet ranges and VLAN IDs. This section should remain unchanged as the Heat template inherits values from its parent template. However, you can modify the values for some parameters using environment files.

Parameter	Description	Type
`ControlPlaneIp`	The node's IP address and subnet on the Control Plane/Provisioning network	string
`ExternalIpSubnet`	The node's IP address and subnet on the External network	string
`InternalApiIpSubnet`	The node's IP address and subnet on the Internal API network	string
`StorageIpSubnet`	The node's IP address and subnet on the Storage network	string
`StorageMgmtIpSubnet`	The node's IP address and subnet on the Storage Management network	string
`TenantIpSubnet`	The node's IP address and subnet on the Tenant network	string
`ManagementIpSubnet`	The node's IP address and subnet on the Management network. Only populated when including **`environments/network-management.yaml`**.	string
`ExternalNetworkVlanID`	The node's VLAN ID for External network traffic.	number
`InternalApiNetworkVlanID`	The node's VLAN ID for Internal API network traffic.	number
`StorageNetworkVlanID`	The node's VLAN ID for Storage network traffic.	number
`StorageMgmtNetworkVlanID`	The node's VLAN ID for Storage Management network traffic.	number
`TenantNetworkVlanID`	The node's VLAN ID for Tenant network traffic.	number
`ManagementNetworkVlanID`	The node's VLAN ID for Management network traffic.	number

Parameter	Description	Type
`ControlPlaneDefaultRoute`	The default route of the Control Plane/Provisioning network. Override this in the **`parameter_defaults`** section of an environment file.	string
`ExternalInterfaceDefaultRoute`	The default route for the External network.	string
`ManagementInterfaceDefaultRoute`	The default route of the Management network.	string
`ControlPlaneSubnetCidr`	The subnet CIDR of the Control Plane/Provisioning network. Override this in the **`parameter_defaults`** section of an environment file.	string
`DnsServers`	A list of DNS servers added to resolv.conf. Usually allows a maximum of 2 servers. Override this in the **`parameter_defaults`** section of an environment file.	comma delimited list
`EC2MetadataIp`	The IP address of the EC2 metadata server. Override this in the **`parameter_defaults`** section of an environment file.	string

Resources

The **`resources`** section is where the main network interface configuration occurs. In most cases, the **`resources`** section is the only one that requires editing. Each **`resources`** section begins with the following header:

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: ../../scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
```

This runs a script (**run-os-net-config.sh**) that creates a configuration file for **os-net-config** to use for configuring network properties on a node. The **network_config** section contains the custom network interface data sent to the **run-os-net-config.sh** script. You arrange this custom interface data in a sequence based on the type of device, which includes the following:

interface

Defines a single network interface. The configuration defines each interface using either the actual interface name ("eth0", "eth1", "enp0s25") or a set of numbered interfaces ("nic1", "nic2", "nic3").

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

vlan

Defines a VLAN. Use the VLAN ID and subnet passed from the **parameters** section.

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

ovs_bond

Defines a bond in Open vSwitch to join two or more **interfaces** together. This helps with redundancy and increases bandwidth.

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

ovs_bridge

Defines a bridge in Open vSwitch, which connects multiple **interface**, **ovs_bond**, and **vlan** objects together. The external bridge also uses two special values for parameters:

- **bridge_name**, which is replaced with the external bridge name.
- **interface_name**, which is replaced with the external interface.

```
- 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}
```

NOTE

The OVS bridge connects to the Neutron server in order to get configuration data. If the OpenStack control traffic (typically the Control Plane and Internal API networks) is placed on an OVS bridge, then connectivity to the Neutron server gets lost whenever OVS is upgraded or the OVS bridge is restarted by the admin user or process. This will cause some downtime. If downtime is not acceptable under these circumstances, then the Control group networks should be placed on a separate interface or bond rather than on an OVS bridge:

- A minimal setting can be achieved, when you put the Internal API network on a VLAN on the provisioning interface and the OVS bridge on a second interface.
- If you want bonding, you need at least two bonds (four network interfaces). The control group should be placed on a Linux bond (Linux bridge). If the switch does not support LACP fallback to a single interface for PXE boot, then this solution requires at least five NICs.

linux_bond

Defines a Linux bond that joins two or more **interfaces** together. This helps with redundancy and increases bandwidth. Make sure to include the kernel-based bonding options in the **bonding_options** parameter. For more information on Linux bonding options, see4.5.1. Bonding Module Directives in the Red Hat Enterprise Linux 7 Networking Guide.

```
              - type: linux_bond
                name: bond1
                members:
                - type: interface
                  name: nic2
                - type: interface
                  name: nic3
                bonding_options: "mode=802.3ad"
```

linux_bridge

Defines a Linux bridge, which connects multiple **interface**, **linux_bond**, and **vlan** objects together. The external bridge also uses two special values for parameters:

- **bridge_name**, which is replaced with the external bridge name.
- **interface_name**, which is replaced with the external interface.

```
              - 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}
```

See Chapter 22, *Network Interface Parameters* for a full list of parameters for each of these items.

The following settings are based on default controller template from the **`/home/stack/templates/nic-configs/controller.yaml`** file. The networks (**`network-config`**) were configured according to the previous recommendations to keep the control group apart from the OVS bridge:

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: ../../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
                  device: nic3
                  vlan_id:
                    get_param: StorageNetworkVlanID
                  addresses:
                  - ip_netmask:
                      get_param: StorageIpSubnet
                - type: vlan
                  device: nic3
                  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}
```

NOTE

The Management network section is commented in the network interface Heat templates. Uncomment this section to enable the Management network.

This template uses four network interfaces and assigns a number of tagged VLAN devices to the numbered interfaces, **nic1** to **nic4**. On **nic3** it creates the OVS bridge that hosts the Storage and Tenant networks.

For more examples of network interface templates, see Appendix B, *Network Interface Template Examples*.

IMPORTANT

Unused interfaces can cause unwanted default routes and network loops. For example, your template might contain a network interface (**nic4**) that does not use any IP assignments for OpenStack services but still uses DHCP and/or a default route. To avoid network conflicts, remove any unused interfaces from **ovs_bridge** devices and disable the DHCP and default route settings:

```
- type: interface
  name: nic4
  use_dhcp: false
  defroute: false
```

8.2. CREATING A NETWORK ENVIRONMENT FILE

The network environment file is a Heat environment file that describes the Overcloud's network environment and points to the network interface configuration templates from the previous section. You can define the subnets and VLANs for your network along with IP address ranges. You can then customize these values for the local environment.

The director contains a set of example environment files to get you started. Each environment file corresponds to the example network interface files in **/usr/share/openstack-tripleo-heat-templates/network/config/**:

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml** - Example environment file for single NIC with VLANs configuration in the **single-nic-vlans**) network interface directory. Environment files for disabling the External network (**net-single-nic-with-vlans-no-external.yaml**) or enabling IPv6 (**net-single-nic-with-vlans-v6.yaml**) are also available.

- **/usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml** - Example environment file for bonded NIC configuration in the **bond-with-vlans** network interface directory. Environment files for disabling the External network (**net-bond-with-vlans-no-external.yaml**) or enabling IPv6 (**net-bond-with-vlans-v6.yaml**) are also available.

- **/usr/share/openstack-tripleo-heat-templates/environments/net-multiple-**

nics.yaml - Example environment file for a multiple NIC configuration in the **multiple-nics** network interface directory. An environment file for enabling IPv6 (**net-multiple-nics-v6.yaml**) is also available.

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-linux-bridge-with-vlans.yaml** - Example environment file for single NIC with VLANs configuration using a Linux bridge instead of an Open vSwitch bridge, which uses the the **single-nic-linux-bridge-vlans** network interface directory.

This scenario uses a modified version of the **/usr/share/openstack-tripleo-heat-templates/environments/net-multiple-nics.yaml** file. Copy this file to the stack user's **templates** directory.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/net-
multiple-nics.yaml /home/stack/templates/network-environment.yaml
```

The environment file contains the following modified sections:

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

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ManagementNetCidr: 172.20.0.0/24
  ExternalNetCidr: 10.1.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end':
'172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end':
'172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end':
'172.19.0.200'}]
  ManagementAllocationPools: [{'start': '172.20.0.10', 'end':
'172.20.0.200'}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{'start': '10.1.1.10', 'end': '10.1.1.50'}]
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 10.1.1.1
  # 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"]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ManagementNetworkVlanID: 205
  ExternalNetworkVlanID: 100
  NeutronExternalNetworkBridge: "''"
```

The **resource_registry** section contains modified links to the custom network interface templates for each node role. Also include links to network interface template for custom roles in this section using the following format:

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

Replace **[ROLE]** with the role name and **[FILE]** with the network interface template location.

The **parameter_defaults** section contains a list of parameters that define the network options for each network type. For a full reference of these options, see Appendix A, *Network Environment Options*.

This scenario defines options for each network. All network types use an individual VLAN and subnet used for assigning IP addresses to hosts and virtual IPs. In the example above, the allocation pool for the Internal API network starts at 172.16.0.10 and continues to 172.16.0.200 using VLAN 201. This results in static and virtual IPs assigned starting at 172.16.0.10 and upwards to 172.16.0.200 while using VLAN 201 in your environment.

The External network hosts the Horizon dashboard and Public API. If using the External network for both cloud administration and floating IPs, make sure there is room for a pool of IPs to use as floating IPs for VM instances. In this example, you only have IPs from 10.1.1.10 to 10.1.1.50 assigned to the External network, which leaves IP addresses from 10.1.1.51 and above free to use for Floating IP addresses. Alternately, place the Floating IP network on a separate VLAN and configure the Overcloud after creation to use it.

If using bonded OVS interfaces, you can configure additional options with **BondInterfaceOvsOptions**. See Appendix C, *Open vSwitch Bonding Options* for more information.

IMPORTANT

Changing the network configuration after creating the Overcloud can cause configuration problems due to the availability of resources. For example, if a user changes a subnet range for a network in the network isolation templates, the reconfiguration might fail due to the subnet already being in use.

8.3. ASSIGNING OPENSTACK SERVICES TO ISOLATED NETWORKS

Each OpenStack service is assigned to a default network type in the resource registry. These services are then bound to IP addresses within the network type’s assigned network. Although the OpenStack services are divided among these networks, the number of actual physical networks might differ as defined in the network environment file. You can reassign OpenStack services to different network types by defining a new network map in your network environment file (**/home/stack/templates/network-environment.yaml**). The **ServiceNetMap** parameter determines the network types used for each service.

For example, you can reassign the Storage Management network services to the Storage Network by modifying the highlighted sections:

```
parameter_defaults:
  ServiceNetMap:
    SwiftMgmtNetwork: storage # Changed from storage_mgmt
    CephClusterNetwork: storage # Changed from storage_mgmt
```

Changing these parameters to **storage** places these services on the Storage network instead of the Storage Management network. This means you only need to define a set of **parameter_defaults** for the Storage network and not the Storage Management network.

The director merges your custom **ServiceNetMap** parameter definitions into a pre-defined list of defaults taken from **ServiceNetMapDefaults** and overrides the defaults. The director then returns the full list including customizations back to **ServiceNetMap**, which is used to configure network assignments for various services.

NOTE

A full list of default services can be found in the **ServiceNetMapDefaults** parameter within **/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml**.

8.4. SELECTING NETWORKS TO DEPLOY

The settings in the **resource_registry** section of the environment file for networks and ports do not ordinarily need to be changed. The list of networks can be changed if only a subset of the networks are desired.

NOTE

When specifying custom networks and ports, do not include the **environments/network-isolation.yaml** on the deployment command line. Instead, specify all the networks and ports in the network environment file.

In order to use isolated networks, the servers must have IP addresses on each network. You can use neutron in the Undercloud to manage IP addresses on the isolated networks, so you will need to enable neutron port creation for each network. You can override the resource registry in your environment file.

First, this is the complete set of the default networks and ports per role that can be deployed:

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
templates/network/storage_mgmt.yaml
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
```

```
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-templates/network/tenant.yaml
  OS::TripleO::Network::Management: /usr/share/openstack-tripleo-heat-templates/network/management.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::ManagementVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/management.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the ceph storage role
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::CephStorage::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the swift storage role
  OS::TripleO::SwiftStorage::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::SwiftStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::SwiftStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the block storage role
  OS::TripleO::BlockStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::BlockStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::BlockStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
```

The first section of this file has the resource registry declaration for the **`OS::TripleO::Network::*`** resources. By default these resources use the **`OS::Heat::None`** resource type, which does not create any networks. By redirecting these resources to the YAML files for each network, you enable the creation of these networks.

The next several sections create the IP addresses for the nodes in each role. The controller nodes have IPs on each network. The compute and storage nodes each have IPs on a subset of the networks.

The default file only contains the port assignments for the default roles. To configure port assignments for a custom role, use the same convention as the other resource definitions and link to the appropriate Heat templates in the **`network/ports`** directory:

- **`OS::TripleO::[ROLE]::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-templates/network/ports/external.yaml`**
- **`OS::TripleO::[ROLE]::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-templates/network/ports/internal_api.yaml`**
- **`OS::TripleO::[ROLE]::Ports::StoragePort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage.yaml`**
- **`OS::TripleO::[ROLE]::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-templates/network/ports/storage_mgmt.yaml`**
- **`OS::TripleO::[ROLE]::Ports::TenantPort: /usr/share/openstack-tripleo-heat-templates/network/ports/tenant.yaml`**
- **`OS::TripleO::[ROLE]::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-templates/network/ports/management.yaml`**

Replace **`[ROLE]`** with the name of your role.

To deploy without one of the pre-configured networks, disable the network definition and the corresponding port definition for the role. For example, all references to **`storage_mgmt.yaml`** could be replaced with **`OS::Heat::None`**:

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
```

```
defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: OS::Heat::None
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: OS::Heat::None
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: OS::Heat::None
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-
heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/tenant.yaml

  # Port assignments for the ceph storage role
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: OS::Heat::None

  # Port assignments for the swift storage role
  OS::TripleO::SwiftStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::SwiftStorage::Ports::StorageMgmtPort: OS::Heat::None
  # Port assignments for the block storage role
  OS::TripleO::BlockStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::BlockStorage::Ports::StorageMgmtPort: OS::Heat::None

parameter_defaults:
  ServiceNetMap:
    ApacheNetwork: internal_api
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    ContrailAnalyticsNetwork: internal_api
    ContrailAnalyticsDatabaseNetwork: internal_api
    ContrailConfigNetwork: internal_api
    ContrailControlNetwork: internal_api
    ContrailDatabaseNetwork: internal_api
    ContrailWebuiNetwork: internal_api
    ContrailTsnNetwork: internal_api
    AodhApiNetwork: internal_api
    PankoApiNetwork: internal_api
    BarbicanApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongodbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    CongressApiNetwork: internal_api
    GlanceApiNetwork: internal_api
    IronicApiNetwork: ctlplane
    IronicNetwork: ctlplane
    IronicInspectorNetwork: ctlplane
    KeystoneAdminApiNetwork: ctlplane # allows undercloud to config
endpoints
    KeystonePublicApiNetwork: internal_api
    ManilaApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    OctaviaApiNetwork: internal_api
    HeatApiNetwork: internal_api
    HeatApiCfnNetwork: internal_api
    HeatApiCloudwatchNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaColdMigrationNetwork: ctlplane
    NovaPlacementNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    NovaLibvirtNetwork: internal_api
    Ec2ApiNetwork: internal_api
    Ec2ApiMetadataNetwork: internal_api
    TackerApiNetwork: internal_api
    SwiftStorageNetwork: storage # Changed from storage_mgmt
    SwiftProxyNetwork: storage
    SaharaApiNetwork: internal_api
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitmqNetwork: internal_api
    QdrNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage # Changed from storage_mgmt
    CephMonNetwork: storage
    CephRgwNetwork: storage
    PublicNetwork: external
    OpendaylightApiNetwork: internal_api
    OvnDbsNetwork: internal_api
    MistralApiNetwork: internal_api
    ZaqarApiNetwork: internal_api
    PacemakerRemoteNetwork: internal_api
    EtcdNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
```

By using **`OS::Heat::None`**, no network or ports are created, so the services on the Storage Management network would default to the Provisioning network. This can be changed in the **`ServiceNetMap`** in order to move the Storage Management services to another network, such as the Storage network.

CHAPTER 9. USING COMPOSABLE NETWORKS

With composable networks, you are no longer constrained by the pre-defined network segments (Internal, Storage, Storage Management, Tenant, External, Control Plane), and instead you can now create your own networks and assign them to any role: default or custom. For example, if you have a network dedicated to NFS traffic, you can now present it to multiple different roles.

Director supports the creation of custom networks during the deployment and update phases. These additional networks can be used for ironic bare metal nodes, system management, or to create separate networks for different roles. They can also be used to create multiple sets of networks for split deployments, where traffic is routed between networks.

A single data file (**`network_data.yaml`**) manages the list of networks that will be deployed; the role definition process then assigns the networks to the required roles through network isolation (see Chapter 8, *Isolating Networks* for more information).

9.1. DEFINING A COMPOSABLE NETWORK

To create composable networks, edit a local copy of the **`/usr/share/openstack-tripleo-heat-templates/network_data.yaml`** Heat template. For example:

```
- 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'
  ipv6_subnet: 'fd00:fd00:fd00:7000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:7000::10', 'end':
'fd00:fd00:fd00:7000:ffff:ffff:ffff:fffe'}]
  gateway_ipv6: 'fd00:fd00:fd00:7000::1'
```

- *name* - is the only mandatory value, however you can also use **`name_lower`** to normalize names for readability. For example, changing **`InternalApi`** to **`internal_api`**.
- *vip:true* will create a virtual IP address (VIP) on the new network, with the remaining parameters setting the defaults for the new network.
- *ip_subnet* and *allocation_pools* will set the default IPv4 subnet and IP range for the network.
- *ipv6_subnet* and *ipv6_allocation_pools* will set the default IPv6 subnets for the network.

NOTE

You can override these defaults using an environment file (usually named *network-environment.yaml*). The sample *network-environment.yaml* file can be created after modifying the network_data.yaml file by running this command from the root of the director's core Heat templates you are using (local copy of */usr/share/openstack-tripleo-heat-templates/*):

```
[stack@undercloud ~/templates] $ ./tools/process-templates.py
```

9.1.1. Define Network Interface Configuration for Composable Networks

When using composable networks, the parameter definition for the network IP address must be added to the NIC configuration template used for each role, even if the network is not used on the role. See the directories in **`/usr/share/openstack-tripleo-heat-templates/network/config`** for examples of these NIC configurations. For instance, if a **`StorageBackup`** network is added to only the Ceph nodes, the following would need to be added to the resource definitions in the NIC configuration templates for all roles:

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

You may also create resource definitions for VLAN IDs and/or gateway IP, if needed:

```
  StorageBackupNetworkVlanID: # Override this via parameter_defaults in
network_environment.yaml
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  StorageBackupDefaultRoute: # Override this via parameter_defaults in
network_environment.yaml
  description: The default route of the storage backup network.
  type: string
```

The **`IpSubnet`** parameter for the custom network appears in the parameter definitions for each role. However, since the Ceph role is the only role that makes use of the **`StorageBackup`** network in our example, only the NIC configuration template for the Ceph role would make use of the **`StorageBackup`** parameters in the **`network_config`** section of the template.

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

9.1.2. Assign Composable Networks to Services

If **`vip: true`** is specified in the custom network definition, then it is possible to assign services to the network using the **`ServiceNetMap`** parameters. The custom network chosen for the service must exist on the role hosting the service. You can override the default networks by overriding the **`ServiceNetMap`** that is defined in **`/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml`** in your **`network_environment.yaml`** (or in a different environment file):

```
parameter_defaults:
  ServiceNetMap:
  NeutronTenantNetwork: tenant
  CeilometerApiNetwork: internal_api
  AodhApiNetwork: internal_api
  GnocchiApiNetwork: internal_api
```

```
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: ctlplane # Admin connection for Undercloud
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage_backup # Changed from storage_mgmt
    SwiftProxyNetwork: storage
    SaharaApiNetwork: internal_api
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage_backup # Changed from storage_mgmt
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

9.1.3. Define the Routed Networks

When using composable networks to deploy routed networks, you define routes and router gateways for use in the network configuration. You can create network routes and supernet routes to define which interface to use when routing traffic between subnets. For example, in a deployment where traffic is routed between the Compute and Controller roles, you may want to define supernets for sets of isolated networks. For instance, **`172.17.0.0/16`** is a supernet that contains all networks beginning with **`172.17`**, so the *Internal API* network used on the controllers might use **`172.17.1.0/24`** and the *Internal API* network used on the Compute nodes might use **`172.17.2.0/24`**. On both roles, you would define a route to the **`172.17.0.0/16`** supernet through the router gateway that is specific to the network used on the role.

The available parameters in **`network-environment.yaml`**:

```
    InternalApiSupernet:
      default: '172.17.0.0/16'
      description: Supernet that contains Internal API subnets for all
  roles.
      type: string
    InternalApiGateway:
      default: '172.17.1.1'
      description: Router gateway on Internal API network
      type: string
    InternalApi2Gateway:
      default: '172.17.2.1'
      description: Router gateway on Internal API 2 network
      Type: string
```

These parameters can be used in the NIC configuration templates for the roles.

The controller uses the parameters for the *InternalApi* network in **`controller.yaml`**:

```
- type: interface
  name: nic3
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
  - routes:
      ip_netmask:
        get_param: InternalApiSupernet
      next_hop:
        Get_param: InternalApiGateway
```

The compute role uses the parameters for the *InternalApi2* network in **`compute.yaml`**:

```
- type: interface
  name: nic3
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: InternalApi2IpSubnet
  - routes:
      ip_netmask:
        get_param: InternalApiSupernet
      next_hop:
        Get_param: InternalApi2Gateway
```

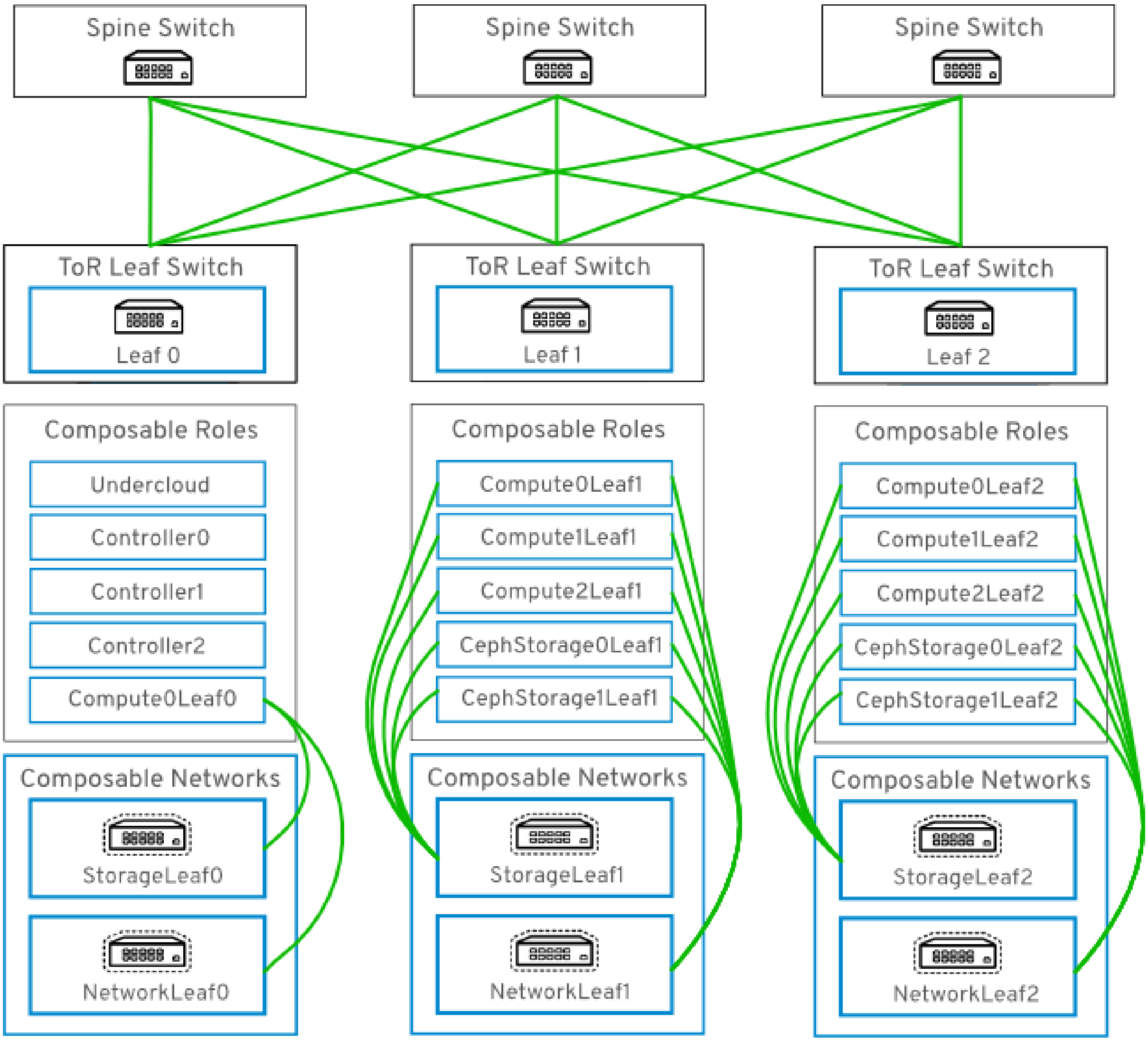
NOTE

If specific network routes are not applied on isolated networks, all traffic to non-local networks use the default gateway. This is generally undesirable from both a security and performance standpoint since it mixes different kinds of traffic and puts all outbound traffic on the same interface. In addition, if the routing is asymmetric (traffic is sent through a different interface than received), it might cause unreachable services. Using a route to the supernet on both the client and server directs traffic to use the correct interface on both sides.

9.2. NETWORKING WITH ROUTED SPINE-LEAF

Composable networks allow you to adapt your OpenStack Networking deployment to the popular routed spine-leaf data center topology. In a practical application of routed spine-leaf, a leaf is represented as a composable Compute or Storage role usually in a datacenter rack, as shown in Figure 9.1, “Routed spine-leaf example”. The *leaf 0* rack has an undercloud node, controllers, and compute nodes. The composable networks are presented to the nodes, which have been assigned to composable roles. In this diagram, the **`StorageLeaf`** networks are presented to the Ceph storage and Compute nodes; the **`NetworkLeaf`** represents an example of any network you may want to compose.

Figure 9.1. Routed spine-leaf example



9.3. HARDWARE PROVISIONING WITH ROUTED SPINE-LEAF

This section describes an example hardware provisioning use case and explains how to deploy an evaluation environment to demonstrate the functionality of routed spine-leaf with composable networks. The resulting deployment has multiple sets of networks with routing available.

To use a provisioning network in a routed spine-leaf network, there are two options available: a VXLAN tunnel configured in the switch fabric, or an extended VLAN trunked to each ToR switch:

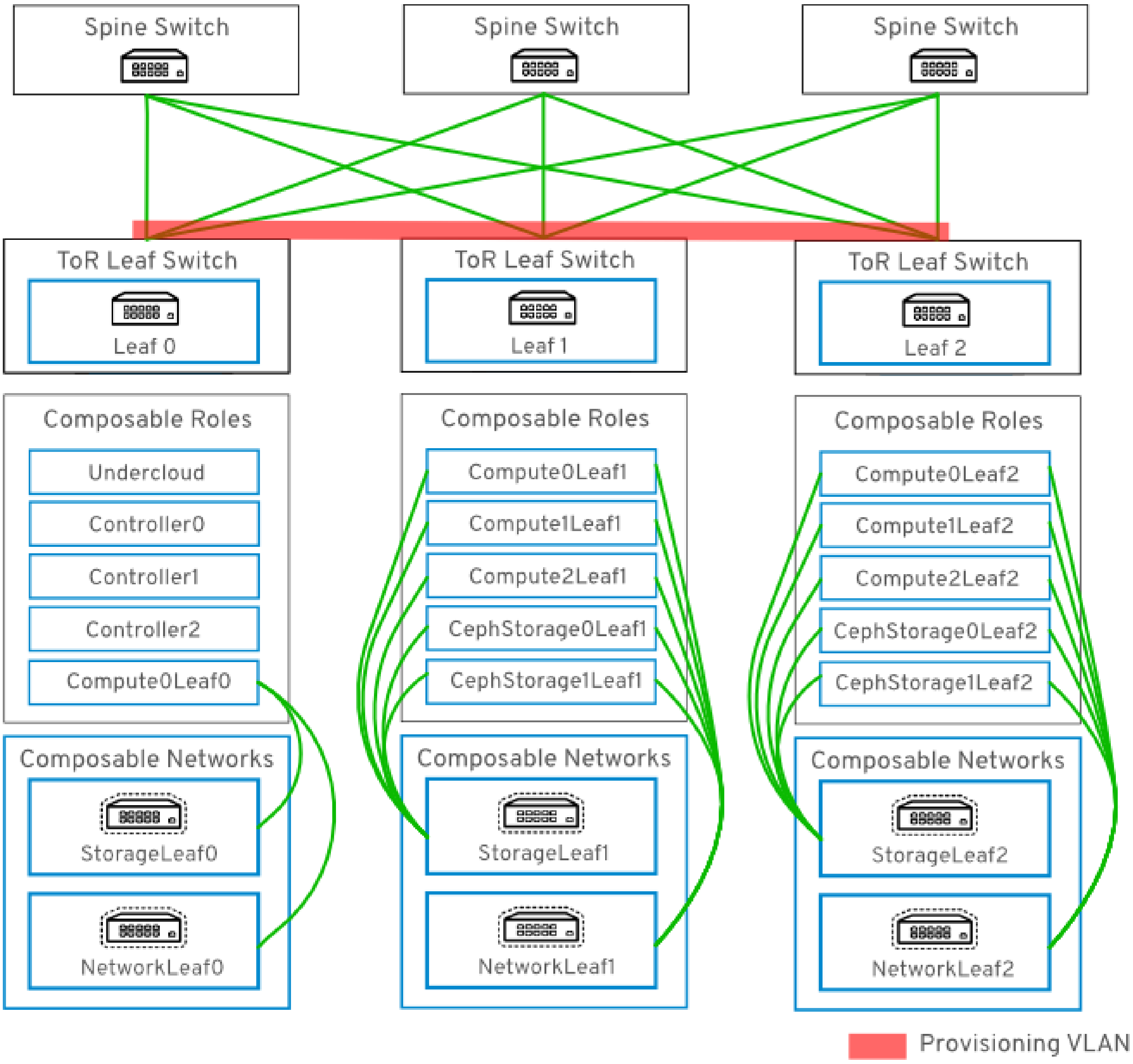
NOTE

In a future release, it is expected that DHCP relays can be used to make **DHCPOFFER** broadcasts traverse across the routed layer 3 domains.

9.3.1. Example VLAN Provisioning Network

In this example, new overcloud nodes are deployed through the provisioning network. The provisioning network cannot be composed, and there cannot be more than one. Instead, a VLAN tunnel is used to span across the layer 3 topology (see Figure 9.2, “VLAN provisioning network topology”). This allows **DHCPOFFER** broadcasts to be sent to any leaf. This tunnel is established by trunking a VLAN between the Top-of-Rack (ToR) leaf switches. In this diagram, the **StorageLeaf** networks are presented to the Ceph storage and Compute nodes; the **NetworkLeaf** represents an example of any network you may want to compose.

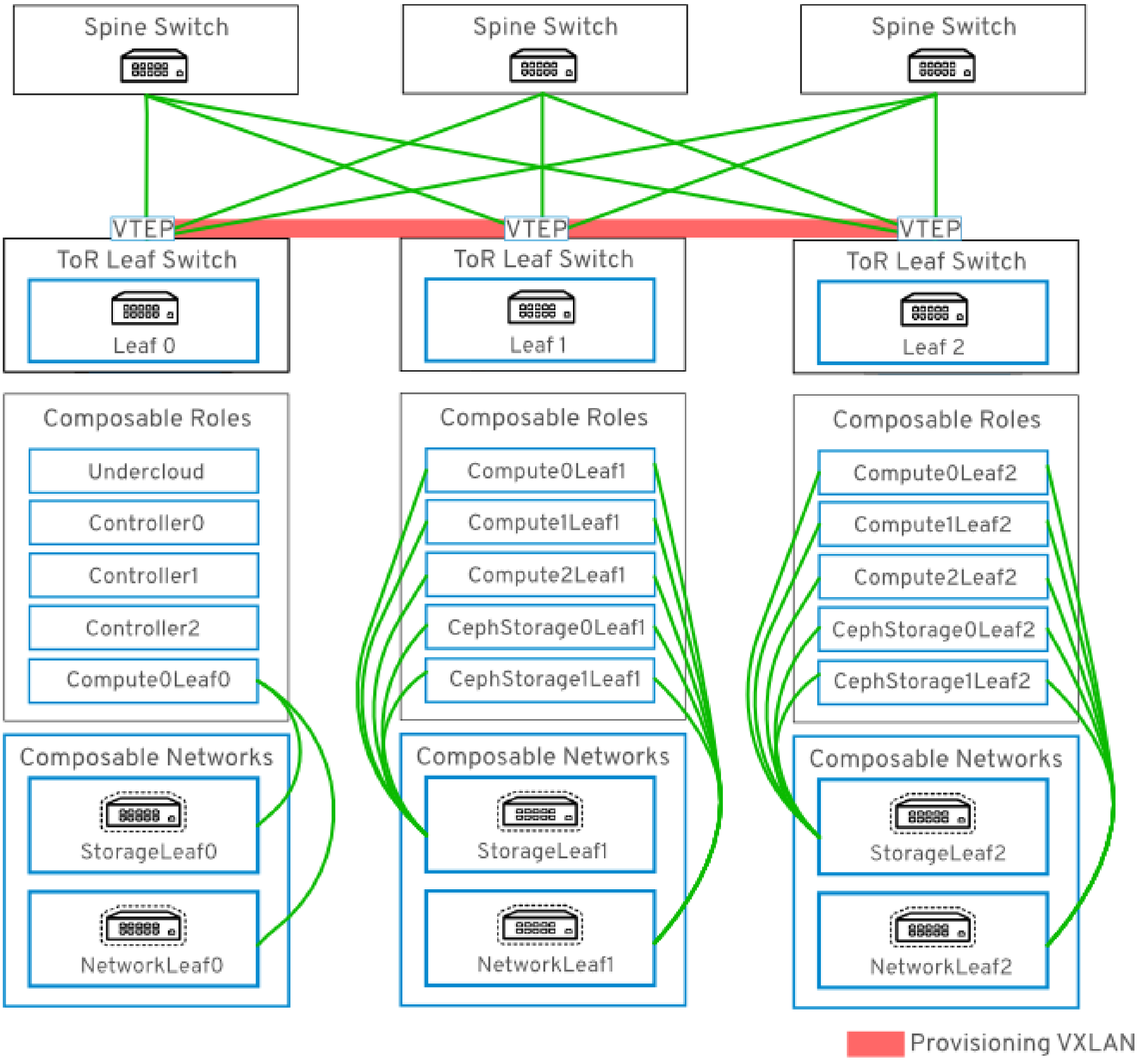
Figure 9.2. VLAN provisioning network topology



9.3.2. Example VXLAN Provisioning Network

In this example, new overcloud nodes are deployed through the provisioning network. The provisioning network cannot be composed, and there cannot be more than one. Instead, VXLAN tunnel is used to span across the layer 3 topology (see Figure 9.3, “VXLAN provisioning network topology”). This allows **DHCPOFFER** broadcasts to be sent to any leaf. This tunnel is established using VXLAN endpoints configured on the Top-of-Rack (ToR) leaf switches.

Figure 9.3. VXLAN provisioning network topology



9.3.3. Network Topology for Provisioning

The routed spine-leaf bare metal environment has one or more layer 3 capable switches, which route traffic between the isolated VLANs in the separate layer 2 broadcast domains.

The intention of this design is to isolate the traffic according to function. For example, if the controller nodes host an API on the *Internal API* network, when a compute node accesses the API it should use its own version of the *Internal API* network. For this routing to work, you need routes that force traffic destined for the *Internal API* network to use the required interface. This can be configured using *supernet* routes. For example, if you use **172.18.0.0/24** as the *Internal API* network for the controller nodes, you can use **172.18.1.0/24** for the second *Internal API* network, and **172.18.2.0/24** for the third, and so on. As a result, you can have a route pointing to the larger **172.18.0.0/16** supernet that uses the gateway IP on the local *Internal API* network for each role in each layer 2 domain.

The following networks could be used in an environment that was deployed using director:

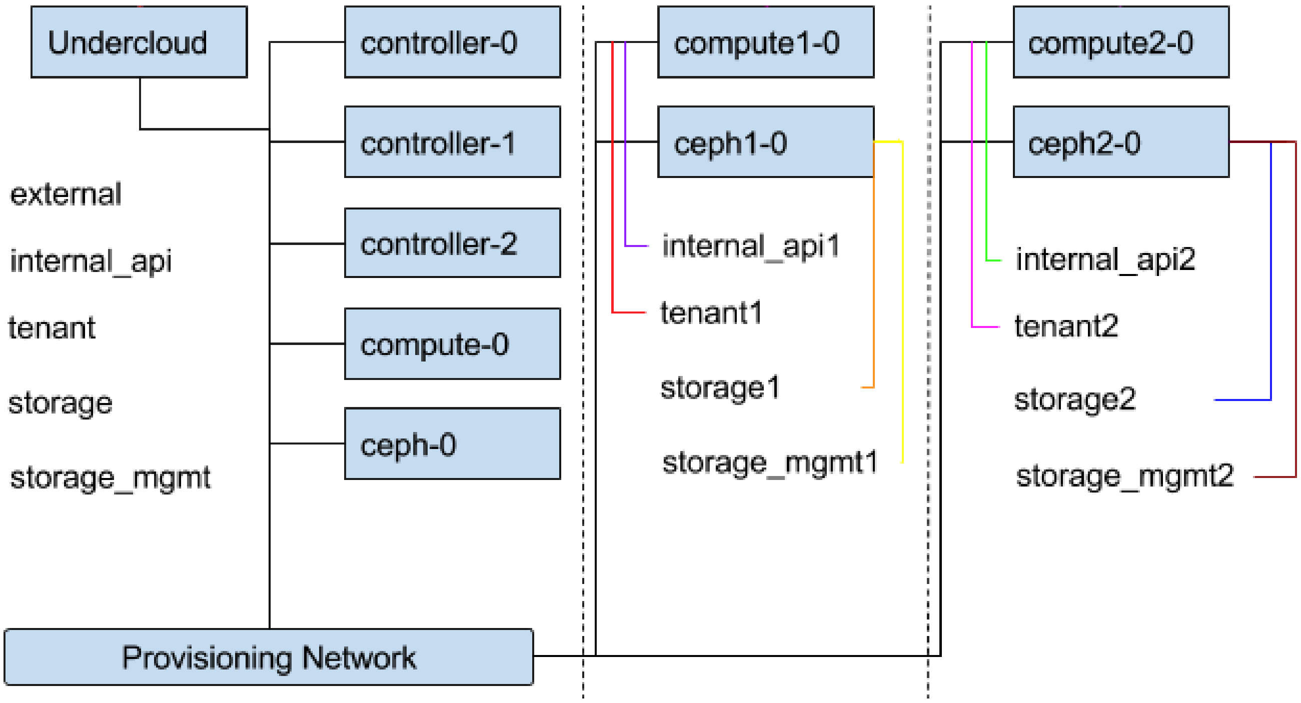
Network	Roles attached	Interface	Bridge	Subnet
Provisioning	All	UC - nic2 and Other - nic1	UC: br-ctlplane	
External	Controller	nic7, OC: nic6	br-ex	192.168.24.0/24
Storage	Controller	nic3, OC: nic2		172.16.0.0/24
Storage Mgmt	Controller	nic4, OC: nic3		172.17.0.0/24
Internal API	Controller	nic5, OC: nic4		172.18.0.0/24
Tenant	Controller	nic6, OC: nic5		172.19.0.0/24
Storage1	Compute1, Ceph1	nic8, OC: nic7		172.16.1.0/24
Storage Mgmt1	Ceph1	nic9, OC: nic8		172.17.1.0/24
Internal API1	Compute1	nic10, OC: nic9		172.18.1.0/24
Tenant1	Compute1	nic11, OC: nic10		172.19.1.0/24
Storage2	Compute2, Ceph2	nic12, OC: nic11		172.16.2.0/24
Storage Mgmt2	Ceph2	nic13, OC: nic12		172.17.2.0/24
Internal API2	Compute2	nic14, OC: nic13		172.18.2.0/24
Tenant2	Compute2	nic15, OC:nic14		172.19.2.0/24

NOTE

The undercloud must also be attached to an uplink for external/Internet connectivity. Typically, the undercloud would be the only node attached to the uplink network. This is likely to be an infrastructure VLAN, separate from the OpenStack deployment.

9.3.4. Topology Diagram

Figure 9.4. Composable Network Topology



9.3.5. Assign IP Addresses to the Custom Roles

The roles require routes for each of the isolated networks. Each role has its own NIC configs and you have to customize the TCP/IP settings to support the custom networks. You can also parameterize or hard-code the gateway IP addresses and routes into the role NIC configs.

For example, using the existing NIC configs as a basic template, you must add the network-specific parameters to all NIC configs:

```
StorageMgmtIpSubnet:
default: ''
description: IP address/subnet on the storage_mgmt network
type: string
StorageMgmt2IpSubnet:
  default: ''
description: IP address/subnet on the storage_mgmt2 network
type: string
TenantIpSubnet:
default: ''
description: IP address/subnet on the tenant network
type: string
TenantIp2Subnet:
default: ''
description: IP address/subnet on the tenant2 network
type: string
```

Perform this for each of the custom networks, for each role used in the deployment.

9.3.6. Assign Routes for the Roles

Each isolated network should have a supernet route applied. Using the suggestion above of **`172.18.0.0/16`** as the supernet route, you would apply the same route to each interface, but using the local gateway.

- **`network-environment.yaml`**:

```
parameter_defaults:
  InternalApiSupernet: 172.18.0.0/16
  InternalApiInterfaceDefaultRoute: 172.18.0.1
  InternalApi1InterfaceDefaultRoute: 172.18.1.1
  InternalApi2InterfaceDefaultRoute: 172.18.2.1
  InternalApi3InterfaceDefaultRoute: 172.18.3.1
```

Each role requires routes on each isolated network, pointing to the other subnets used for the same function. So when a *Compute1* node contacts a controller on the **`InternalApi`** VIP, the traffic should target the **`InternalApi1`** interface through the **`InternalApi1`** gateway. As a result, the return traffic from the controller to the **`InternalApi1`** network should go through the **`InternalApi`** network gateway.

- Controller configuration:

```
            - type: interface
              name: nic4
              use_dhcp: false
              addresses:
              - ip_netmask:
                      get_param: InternalApiIpSubnet
                 routes:
                   - ip_netmask:
                       get_param: InternalApiSupernet
                     next_hop:
                       get_param: InternalApiDefaultRoute
```

- Compute1 configuration:

```
            - type: interface
              name: nic4
              use_dhcp: false
              addresses:
              - ip_netmask:
                      get_param: InternalApi1IpSubnet
                 routes:
                   - ip_netmask:
                       get_param: InternalApiSupernet
                     next_hop:
                       get_param: InternalApi1DefaultRoute
```

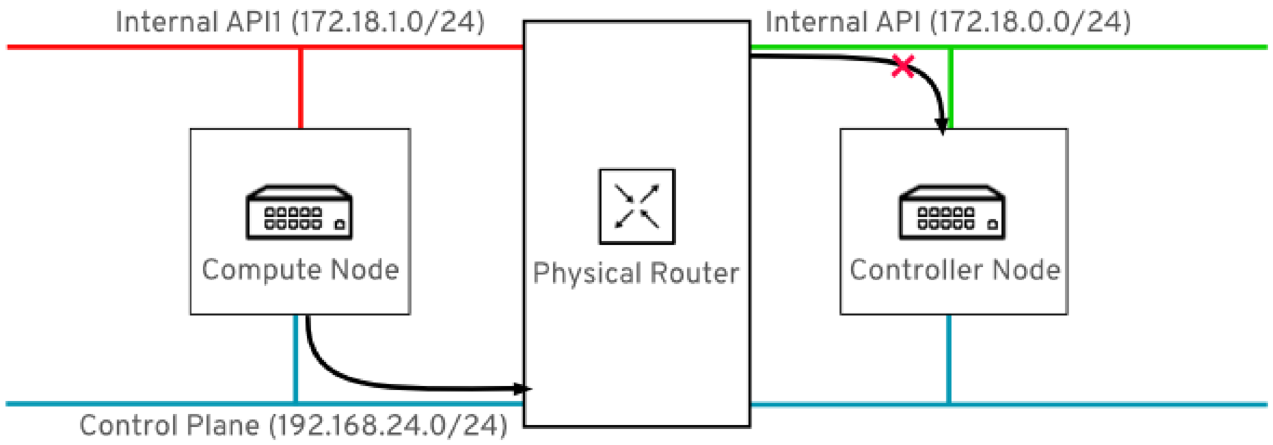
The supernet routes apply to all isolated networks on each role to avoid sending traffic through the default gateway, which by default is the *Control Plane* network on non-controllers, and the *External* network on the controllers.

You need to configure these routes on the isolated networks because Red Hat Enterprise Linux by default implements strict reverse path filtering on inbound traffic. If an API is listening on the *Internal API* interface and a request comes in to that API, it only accepts the request if the return path route is on the

Internal API interface. If the server is listening on the *Internal API* network but the return path to the client is through the *Control Plane*, then the server drops the requests due to the reverse path filter.

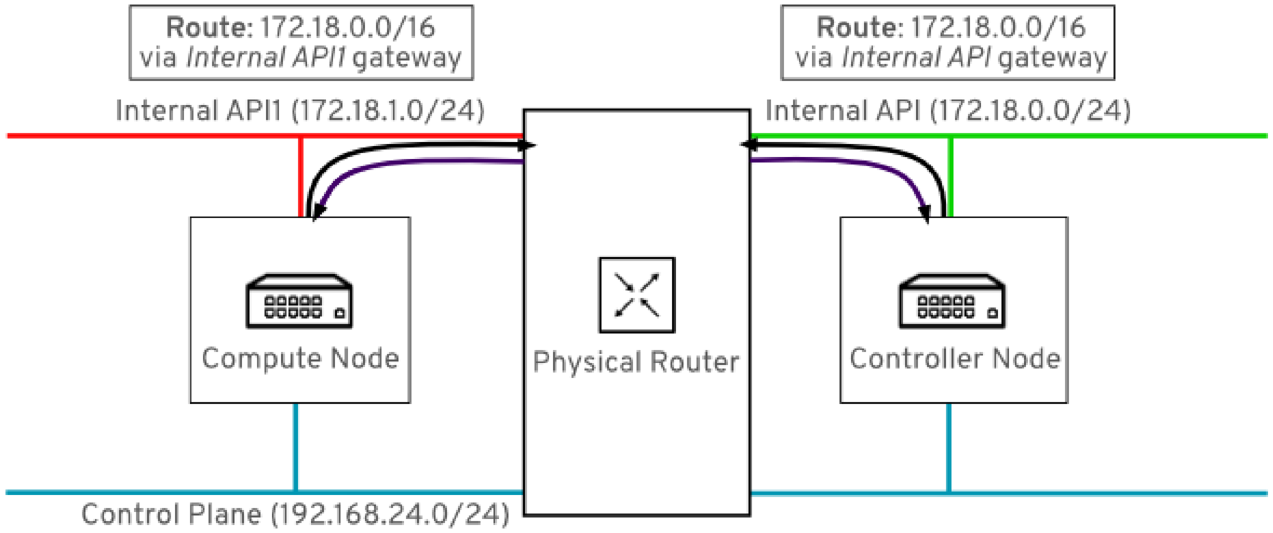
For example, this diagram shows an attempt to route traffic through the control plane, which will not succeed. The return route from the router to the controller node does not match the interface where the VIP is listening, so the packet is dropped. **`192.168.24.0/24`** is directly connected to the controller, so it is considered local to the *Control Plane* network.

Figure 9.5. Routed traffic through Control Plane


Internal API1 (172.18.1.0/24)
Internal API (172.18.0.0/24)
Compute Node
Physical Router
Controller Node
Control Plane (192.168.24.0/24)

For comparison, this diagram shows routing running through the *Internal API* networks:

Figure 9.6. Routed traffic through Internal API


Route: 172.18.0.0/16
via Internal API1 gateway
Route: 172.18.0.0/16
via Internal API gateway
Internal API1 (172.18.1.0/24)
Internal API (172.18.0.0/24)
Compute Node
Physical Router
Controller Node
Control Plane (192.168.24.0/24)

In this diagram, the return route to **`172.18.1.0`** matches the interface where the virtual IP address (VIP) is listening. As a result, packets are not dropped and the API connectivity works as expected.

The following **`ExtraConfig`** settings address the issue described above. Note that the **`InternalApi1`** value is ultimately represented by the **`internal_api1`** value and is case-sensitive.

```
parameter_defaults:
  Compute1ExtraConfig:
     nova::vncproxy::host: "%{hiera('internal_api1')}"
     neutron::agents::ml2::ovs::local_ip: "%{hiera('tenant1')}"
  Compute2ExtraConfig:
     nova::vncproxy::host: "%{hiera('internal_api2')}"
     neutron::agents::ml2::ovs::local_ip: "%{hiera('tenant2')}"
  Compute3ExtraConfig:
     nova::vncproxy::host: "%{hiera('internal_api3')}"
     neutron::agents::ml2::ovs::local_ip: "%{hiera('tenant3')}"
  CephAnsibleExtraConfig:
    public_network:
'172.120.3.0/24,172.117.3.0/24,172.118.3.0/24,172.119.3.0/24'
    cluster_network:
'172.120.4.0/24,172.117.4.0/24,172.118.4.0/24,172.119.4.0/24'
```

- **CephAnsibleExtraConfig** - The **public_network** setting lists all the storage network leaves. The **cluster_network** entries lists the storage management networks (one per leaf).

9.3.7. Custom NIC definitions

The following custom definitions were applied in the **nic-config** template for nodes. Change the following example to suit your deployment:

1. Review the **network_data.yaml** values. They should be similar to the following example:

```
[stack@undercloud-0 ~]$ cat /home/stack/network_data.yaml
- name: External
  vip: true
  name_lower: external
  ip_subnet: '10.0.0.0/24'
  allocation_pools: [{'start': '10.0.0.4', 'end': '10.0.0.250'}]
  gateway_ip: '10.0.0.1'
  ipv6_subnet: '2001:db8:fd00:1000::/64'
  ipv6_allocation_pools: [{'start': '2001:db8:fd00:1000::10', 'end':
'2001:db8:fd00:1000:ffff:ffff:ffff:fffe'}]
  gateway_ipv6: '2001:db8:fd00:1000::1'
- name: InternalApi
  name_lower: internal_api
  vip: true
  ip_subnet: '172.16.2.0/24'
  allocation_pools: [{'start': '172.16.2.4', 'end': '172.16.2.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2000::10',
'end': 'fd00:fd00:fd00:2000:ffff:ffff:ffff:fffe'}]
- name: Storage
  vip: true
  name_lower: storage
  ip_subnet: '172.16.1.0/24'
  allocation_pools: [{'start': '172.16.1.4', 'end': '172.16.1.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:3000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:3000::10',
'end': 'fd00:fd00:fd00:3000:ffff:ffff:ffff:fffe'}]
- name: StorageMgmt
  name_lower: storage_mgmt
  vip: true
  ip_subnet: '172.16.3.0/24'
  allocation_pools: [{'start': '172.16.3.4', 'end': '172.16.3.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:4000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:4000::10',
'end': 'fd00:fd00:fd00:4000:ffff:ffff:ffff:fffe'}]
- name: Tenant
  vip: false  # Tenant network does not use VIPs
  name_lower: tenant
  ip_subnet: '172.16.0.0/24'
  allocation_pools: [{'start': '172.16.0.4', 'end': '172.16.0.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:5000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:5000::10',
'end': 'fd00:fd00:fd00:5000:ffff:ffff:ffff:fffe'}]
- name: Management
  # Management network is enabled by default for backwards-
compatibility, but
  # is not included in any roles by default. Add to role definitions
to use.
  enabled: true
  vip: false  # Management network does not use VIPs
  name_lower: management
  ip_subnet: '10.0.1.0/24'
  allocation_pools: [{'start': '10.0.1.4', 'end': '10.0.1.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:6000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:6000::10',
'end': 'fd00:fd00:fd00:6000:ffff:ffff:ffff:fffe'}]
- name: Tenant1
  vip: false  # Tenant network does not use VIPs
  name_lower: tenant1
  ip_subnet: '172.16.11.0/24'
  allocation_pools: [{'start': '172.16.11.4', 'end':
'172.16.11.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:5001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:5001::10',
'end': 'fd00:fd00:fd00:5001:ffff:ffff:ffff:fffe'}]
- name: Tenant2
  vip: false  # Tenant network does not use VIPs
  name_lower: tenant2
  ip_subnet: '172.16.12.0/24'
  allocation_pools: [{'start': '172.16.12.4', 'end':
'172.16.12.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:5002::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:5002::10',
'end': 'fd00:fd00:fd00:5002:ffff:ffff:ffff:fffe'}]
- name: Tenant3
  vip: false  # Tenant network does not use VIPs
  name_lower: tenant3
  ip_subnet: '172.16.13.0/24'
  allocation_pools: [{'start': '172.16.13.4', 'end':
'172.16.13.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:5003::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:5003::10',
'end': 'fd00:fd00:fd00:5003:ffff:ffff:ffff:fffe'}]
- name: StorageMgmt1
  name_lower: storage_mgmt1
  vip: true
  ip_subnet: '172.16.21.0/24'
  allocation_pools: [{'start': '172.16.21.4', 'end':
'172.16.21.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:4001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:4001::10',
'end': 'fd00:fd00:fd00:4001:ffff:ffff:ffff:fffe'}]
- name: StorageMgmt2
  name_lower: storage_mgmt2
  vip: true
  ip_subnet: '172.16.22.0/24'
  allocation_pools: [{'start': '172.16.22.4', 'end':
'172.16.22.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:4002::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:4002::10',
'end': 'fd00:fd00:fd00:4002:ffff:ffff:ffff:fffe'}]
- name: StorageMgmt3
  name_lower: storage_mgmt3
  vip: true
  ip_subnet: '172.16.23.0/24'
  allocation_pools: [{'start': '172.16.23.4', 'end':
'172.16.23.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:4003::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:4003::10',
'end': 'fd00:fd00:fd00:4003:ffff:ffff:ffff:fffe'}]
- name: Storage1
  vip: true
  name_lower: storage1
  ip_subnet: '172.16.31.0/24'
  allocation_pools: [{'start': '172.16.31.4', 'end':
'172.16.31.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:3001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:3001::10',
'end': 'fd00:fd00:fd00:3001:ffff:ffff:ffff:fffe'}]
- name: Storage2
  vip: true
  name_lower: storage2
  ip_subnet: '172.16.32.0/24'
  allocation_pools: [{'start': '172.16.32.4', 'end':
'172.16.32.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:3002::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:3002::10',
'end': 'fd00:fd00:fd00:3002:ffff:ffff:ffff:fffe'}]
- name: Storage3
  vip: true
  name_lower: storage3
  ip_subnet: '172.16.33.0/24'
  allocation_pools: [{'start': '172.16.33.4', 'end':
'172.16.33.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:3003::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:3003::10',
'end': 'fd00:fd00:fd00:3003:ffff:ffff:ffff:fffe'}]
- name: InternalApi1
  name_lower: internal_api1
  vip: true
  ip_subnet: '172.16.41.0/24'
  allocation_pools: [{'start': '172.16.41.4', 'end':
'172.16.41.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2001::10',
'end': 'fd00:fd00:fd00:2001:ffff:ffff:ffff:fffe'}]
```

```
- name: InternalApi2
  name_lower: internal_api2
  vip: true
  ip_subnet: '172.16.42.0/24'
  allocation_pools: [{'start': '172.16.42.4', 'end':
'172.16.42.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2002::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2002::10',
'end': 'fd00:fd00:fd00:2002:ffff:ffff:ffff:fffe'}]
- name: InternalApi3
  name_lower: internal_api3
  vip: true
  ip_subnet: '172.16.43.0/24'
  allocation_pools: [{'start': '172.16.43.4', 'end':
'172.16.43.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2003::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2003::10',
'end': 'fd00:fd00:fd00:2003:ffff:ffff:ffff:fffe'}]
```

NOTE

There is currently no validation performed for the network subnet and **`allocation_pools`** values. Be certain you have defined these consistently and there is no conflict with existing networks.

2. Review the **`/home/stack/roles_data.yaml`** values. They should be similar to the following example:

```
####################################
# Role: Controller                 #
####################################
- name: Controller
  description: |
  Controller role that has all the controler services loaded and
handles
  Database, Messaging and Network functions.
  CountDefault: 1
  tags:
  - primary
  - controller
  networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
  HostnameFormatDefault: '%stackname%-controller-%index%'
  ServicesDefault:
  - OS::TripleO::Services::AodhApi
  - OS::TripleO::Services::AodhEvaluator
  - OS::TripleO::Services::AodhListener
  - OS::TripleO::Services::AodhNotifier
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BarbicanApi
  - OS::TripleO::Services::CACerts
```

```
    - OS::TripleO::Services::CeilometerAgentCentral
    - OS::TripleO::Services::CeilometerAgentNotification
    - OS::TripleO::Services::CeilometerApi
    - OS::TripleO::Services::CeilometerCollector
    - OS::TripleO::Services::CeilometerExpirer
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephMds
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephRbdMirror
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackendDellPs
    - OS::TripleO::Services::CinderBackendDellSc
    - OS::TripleO::Services::CinderBackendDellEMCUnity
    - OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
    - OS::TripleO::Services::CinderBackendNetApp
    - OS::TripleO::Services::CinderBackendScaleIO
    - OS::TripleO::Services::CinderBackendVRTSHyperScale
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderHPELeftHandISCSI
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Clustercheck
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Congress
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Ec2Api
    - OS::TripleO::Services::Etcd
    - OS::TripleO::Services::ExternalSwiftProxy
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GnocchiApi
    - 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
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::ManilaApi
    - OS::TripleO::Services::ManilaBackendCephFs
    - OS::TripleO::Services::ManilaBackendGeneric
    - OS::TripleO::Services::ManilaBackendIsilon
    - OS::TripleO::Services::ManilaBackendNetapp
    - OS::TripleO::Services::ManilaBackendUnity
    - OS::TripleO::Services::ManilaBackendVNX
    - OS::TripleO::Services::ManilaBackendVMAX
    - OS::TripleO::Services::ManilaScheduler
    - OS::TripleO::Services::ManilaShare
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::MongoDb
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronBgpVpnApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL2gwAgent
    - OS::TripleO::Services::NeutronL2gwApi
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronLbaasv2Agent
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronML2FujitsuCfab
    - OS::TripleO::Services::NeutronML2FujitsuFossw
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::NovaConsoleauth
    - OS::TripleO::Services::NovaIronic
    - OS::TripleO::Services::NovaMetadata
    - OS::TripleO::Services::NovaPlacement
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaVncProxy
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OctaviaApi
    - OS::TripleO::Services::OctaviaHealthManager
    - OS::TripleO::Services::OctaviaHousekeeping
    - OS::TripleO::Services::OctaviaWorker
    - OS::TripleO::Services::OpenDaylightApi
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::OVNDBs
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::PankoApi
    - OS::TripleO::Services::RabbitMQ
    - OS::TripleO::Services::Redis
    - OS::TripleO::Services::SaharaApi
    - OS::TripleO::Services::SaharaEngine
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::Tacker
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
```

```
  - OS::TripleO::Services::Zaqar
####################################
# Role: Compute                    #
####################################
- name: Compute1
  description: |
  Basic Compute Node role
  CountDefault: 1
  networks:
  - InternalApi1
  - Tenant1
  - Storage1
  HostnameFormatDefault: '%stackname%-novacompute1-%index%'
  disable_upgrade_deployment: True
  ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::ComputeCeilometerAgent
  - OS::TripleO::Services::ComputeNeutronCorePlugin
  - OS::TripleO::Services::ComputeNeutronL3Agent
  - OS::TripleO::Services::ComputeNeutronMetadataAgent
  - OS::TripleO::Services::ComputeNeutronOvsAgent
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::FluentdClient
  - OS::TripleO::Services::Iscsid
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::NeutronLinuxbridgeAgent
  - OS::TripleO::Services::NeutronSriovAgent
  - OS::TripleO::Services::NeutronSriovHostConfig
  - OS::TripleO::Services::NeutronVppAgent
  - OS::TripleO::Services::NovaCompute
  - OS::TripleO::Services::NovaLibvirt
  - OS::TripleO::Services::NovaMigrationTarget
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::OpenDaylightOvs
  - OS::TripleO::Services::Securetty
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Sshd
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
  - OS::TripleO::Services::Vpp
  - OS::TripleO::Services::OVNController
####################################
# Role: CephStorage                #
####################################
- name: CephStorage1
  description: |
  Ceph OSD Storage node role
  networks:
  - Storage1
  - StorageMgmt1
  ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephOSD
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::FluentdClient
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::Securetty
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Sshd
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
####################################
# Role: Compute                    #
####################################
- name: Compute2
  description: |
  Basic Compute Node role
  CountDefault: 1
  networks:
  - InternalApi2
  - Tenant2
  - Storage2
  HostnameFormatDefault: '%stackname%-novacompute2-%index%'
  disable_upgrade_deployment: True
  ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::ComputeCeilometerAgent
  - OS::TripleO::Services::ComputeNeutronCorePlugin
  - OS::TripleO::Services::ComputeNeutronL3Agent
  - OS::TripleO::Services::ComputeNeutronMetadataAgent
  - OS::TripleO::Services::ComputeNeutronOvsAgent
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::FluentdClient
  - OS::TripleO::Services::Iscsid
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::NeutronLinuxbridgeAgent
  - OS::TripleO::Services::NeutronSriovAgent
  - OS::TripleO::Services::NeutronSriovHostConfig
  - OS::TripleO::Services::NeutronVppAgent
  - OS::TripleO::Services::NovaCompute
  - OS::TripleO::Services::NovaLibvirt
  - OS::TripleO::Services::NovaMigrationTarget
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::OpenDaylightOvs
  - OS::TripleO::Services::Securetty
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Sshd
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
  - OS::TripleO::Services::Vpp
  - OS::TripleO::Services::OVNController
####################################
# Role: CephStorage                #
####################################
- name: CephStorage2
  description: |
  Ceph OSD Storage node role
  networks:
  - Storage2
  - StorageMgmt2
  ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephOSD
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::FluentdClient
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::Securetty
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Sshd
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
####################################
# Role: Compute                    #
####################################
- name: Compute3
  description: |
  Basic Compute Node role
  CountDefault: 1
  networks:
  - InternalApi3
  - Tenant3
  - Storage3
  HostnameFormatDefault: '%stackname%-novacompute3-%index%'
  disable_upgrade_deployment: True
  ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::ComputeCeilometerAgent
  - OS::TripleO::Services::ComputeNeutronCorePlugin
  - OS::TripleO::Services::ComputeNeutronL3Agent
  - OS::TripleO::Services::ComputeNeutronMetadataAgent
  - OS::TripleO::Services::ComputeNeutronOvsAgent
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::FluentdClient
  - OS::TripleO::Services::Iscsid
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::NeutronLinuxbridgeAgent
  - OS::TripleO::Services::NeutronSriovAgent
  - OS::TripleO::Services::NeutronSriovHostConfig
  - OS::TripleO::Services::NeutronVppAgent
  - OS::TripleO::Services::NovaCompute
  - OS::TripleO::Services::NovaLibvirt
  - OS::TripleO::Services::NovaMigrationTarget
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::OpenDaylightOvs
  - OS::TripleO::Services::Securetty
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Sshd
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
  - OS::TripleO::Services::Vpp
  - OS::TripleO::Services::OVNController
####################################
# Role: CephStorage                #
####################################
- name: CephStorage3
  description: |
  Ceph OSD Storage node role
  networks:
  - Storage3
  - StorageMgmt3
  ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephOSD
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::FluentdClient
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::Securetty
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Sshd
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
```

3. Review the **`nic-config`** template for the Compute node:

```
[stack@undercloud-0 ~]$ cat virt/network/three-nics-
vlans/compute1.yaml
heat_template_version: 2015-04-30

description: >
  Software Config to drive os-net-config to configure multiple
interfaces
  for the compute role.

parameters:
  InternalApi1InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  InternalApi2InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  InternalApi3InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  Tenant1InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  Tenant2InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  Tenant3InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  Storage1InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  Storage2InterfaceDefaultRoute: # Override this via
```

```
parameter_defaults
  description: Default route for the specific network.
  type: string
  Storage3InterfaceDefaultRoute: # Override this via
parameter_defaults
  description: Default route for the specific network.
  type: string
  InternalApi1NetworkVlanID:
  default: 21
  description: Vlan ID for the internal_api network traffic.
  type: number
  InternalApi2NetworkVlanID:
  default: 22
  description: Vlan ID for the internal_api network traffic.
  type: number
  InternalApi3NetworkVlanID:
  default: 23
  description: Vlan ID for the internal_api network traffic.
  type: number
  Storage1NetworkVlanID:
  default: 31
  description: Vlan ID for the storage network traffic.
  type: number
  Storage2NetworkVlanID:
  default: 32
  description: Vlan ID for the storage network traffic.
  type: number
  Storage3NetworkVlanID:
  default: 33
  description: Vlan ID for the storage network traffic.
  type: number
  StorageMgmt1NetworkVlanID:
  default: 41
  description: Vlan ID for the storage mgmt network traffic.
  type: number
  StorageMgmt2NetworkVlanID:
  default: 42
  description: Vlan ID for the storage mgmt network traffic.
  type: number
  StorageMgmt3NetworkVlanID:
  default: 43
  description: Vlan ID for the storage mgmt network traffic.
  type: number
  Tenant1NetworkVlanID:
  default: 51
  description: Vlan ID for the tenant network traffic.
  type: number
  Tenant2NetworkVlanID:
  default: 52
  description: Vlan ID for the tenant network traffic.
  type: number
  Tenant3NetworkVlanID:
  default: 53
  description: Vlan ID for the tenant network traffic.
  type: number
  ControlPlaneIp:
```

```
default: ''
description: IP address/subnet on the ctlplane network
type: string
ExternalIpSubnet:
default: ''
description: IP address/subnet on the external network
type: string
InternalApiIpSubnet:
default: ''
description: IP address/subnet on the internal API network
type: string
InternalApi1IpSubnet:
default: ''
description: IP address/subnet on the internal API network
type: string
InternalApi2IpSubnet:
default: ''
description: IP address/subnet on the internal API network
type: string
InternalApi3IpSubnet:
default: ''
description: IP address/subnet on the internal API network
type: string
Storage1IpSubnet:
default: ''
description: IP address/subnet on the storage network
type: string
Storage2IpSubnet:
default: ''
description: IP address/subnet on the storage network
type: string
Storage3IpSubnet:
default: ''
description: IP address/subnet on the storage network
type: string
StorageMgmt1IpSubnet:
default: ''
description: IP address/subnet on the storage mgmt network
type: string
StorageMgmt2IpSubnet:
default: ''
description: IP address/subnet on the storage mgmt network
type: string
StorageMgmt3IpSubnet:
default: ''
description: IP address/subnet on the storage mgmt network
type: string
Tenant1IpSubnet:
default: ''
description: IP address/subnet on the tenant network
type: string
Tenant2IpSubnet:
default: ''
description: IP address/subnet on the tenant network
type: string
Tenant3IpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
  StorageIpSubnet:
  default: ''
  description: IP address/subnet on the storage network
  type: string
  StorageMgmtIpSubnet:
  default: ''
  description: IP address/subnet on the storage mgmt network
  type: string
  TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
  ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
  default: ''
  description: IP address/subnet on the management network
  type: string
  InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
  StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
  TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
  ControlPlaneSubnetCidr: # Override this via parameter_defaults
  default: '24'
  description: The subnet CIDR of the control plane network.
  type: string
  ControlPlaneDefaultRoute: # Override this via parameter_defaults
  description: The subnet CIDR of the control plane network.
  type: string
  DnsServers: # Override this via parameter_defaults
  default: []
  description: A list of DNS servers (2 max for some
implementations) that will be added to resolv.conf.
  type: json
  EC2MetadataIp: # Override this via parameter_defaults
  description: The IP address of the EC2 metadata server.
  type: string

resources:
  OsNetConfigImpl:
  type: OS::Heat::StructuredConfig
  properties:
    group: os-apply-config
    config:
      os_net_config:
        network_config:
          -
            type: interface
            name: nic1
            use_dhcp: false
            dns_servers: {get_param: DnsServers}
            addresses:
              -
                ip_netmask:
                  list_join:
                    - '/'
                    - - {get_param: ControlPlaneIp}
                      - {get_param: ControlPlaneSubnetCidr}
            routes:
              -
                ip_netmask: 0.0.0.0/0
                next_hop: {get_param: ControlPlaneDefaultRoute}
                # Optionally have this interface as default route
                default: true
              -
                ip_netmask: 169.254.169.254/32
                next_hop: {get_param: EC2MetadataIp}
          -
            type: ovs_bridge
            name: br-isolated
            use_dhcp: false
            members:
              -
                type: interface
                name: nic2
                # force the MAC address of the bridge to this
interface
                primary: true
              -
                type: vlan
                vlan_id: {get_param: InternalApi1NetworkVlanID}
                addresses:
                -
                  ip_netmask: {get_param: InternalApi1IpSubnet}
                routes:
                -
                  ip_netmask: 172.120.1.0/24
                  next_hop: {get_param:
InternalApi1InterfaceDefaultRoute}
                -
                  ip_netmask: 172.118.1.0/24
                  next_hop: {get_param:
InternalApi1InterfaceDefaultRoute}
                -
                  ip_netmask: 172.119.1.0/24
                  next_hop: {get_param:
InternalApi1InterfaceDefaultRoute}
              -
                type: vlan
                vlan_id: {get_param: Storage1NetworkVlanID}
                addresses:
                -
```

```
                  ip_netmask: {get_param: Storage1IpSubnet}
                routes:
                -
                  ip_netmask: 172.120.3.0/24
                  next_hop: {get_param:
Storage1InterfaceDefaultRoute}
                -
                  ip_netmask: 172.118.3.0/24
                  next_hop: {get_param:
Storage1InterfaceDefaultRoute}
                -
                  ip_netmask: 172.119.3.0/24
                  next_hop: {get_param:
Storage1InterfaceDefaultRoute}
              -
                type: vlan
                vlan_id: {get_param: Tenant1NetworkVlanID}
                addresses:
                -
                  ip_netmask: {get_param: Tenant1IpSubnet}
                routes:
                -
                  ip_netmask: 172.120.2.0/24
                  next_hop: {get_param:
Tenant1InterfaceDefaultRoute}
                -
                  ip_netmask: 172.118.2.0/24
                  next_hop: {get_param:
Tenant1InterfaceDefaultRoute}
                -
                  ip_netmask: 172.119.2.0/24
                  next_hop: {get_param:
Tenant1InterfaceDefaultRoute}
          -
            type: interface
            name: nic3
            use_dhcp: false

outputs:
  OS::stack_id:
  description: The OsNetConfigImpl resource.
  value: {get_resource: OsNetConfigImpl}
```

4. Run the **`openstack overcloud deploy`** command to apply the changes. For example:

```
openstack overcloud deploy --templates \
--libvirt-type kvm \
-n /home/stack/network_data.yaml \
-r /home/stack/roles_data.yaml \
-e /home/stack/templates/nodes_data.yaml \
-e  /usr/share/openstack-tripleo-heat-templates/environments/ceph-
ansible/ceph-ansible.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-
isolation.yaml \
-e /home/stack/virt/network/network-environment.yaml \
-e /usr/share/openstack-tripleo-heat-
```

```
templates/environments/ssl/enable-tls.yaml \
-e /home/stack/virt/public_vip.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-
endpoints-public-ip.yaml \
-e /home/stack/inject-trust-anchor-hiera.yaml \
-e /home/stack/rhos12.yaml
```

CHAPTER 10. CONTROLLING NODE PLACEMENT

The default behavior for the director is to randomly select nodes for each role, usually based on their profile tag. However, the director provides the ability to define specific node placement. This is a useful method to:

- Assign specific node IDs e.g. **controller-0**, **controller-1**, etc
- Assign custom hostnames
- Assign specific IP addresses
- Assign specific Virtual IP addresses

NOTE

Manually setting predictable IP addresses, virtual IP addresses, and ports for a network alleviates the need for allocation pools. However, it is recommended to retain allocation pools for each network to ease with scaling new nodes. Make sure that any statically defined IP addresses fall outside the allocation pools. For more information on setting allocation pools, see Section 8.2, “Creating a Network Environment File”.

10.1. ASSIGNING SPECIFIC NODE IDS

This procedure assigns node ID to specific nodes. Examples of node IDs include **controller-0**, **controller-1**, **compute-0**, **compute-1**, and so forth.

The first step is to assign the ID as a per-node capability that the Nova scheduler matches on deployment. For example:

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

This assigns the capability **node:controller-0** to the node. Repeat this pattern using a unique continuous index, starting from 0, for all nodes. Make sure all nodes for a given role (Controller, Compute, or each of the storage roles) are tagged in the same way or else the Nova scheduler will not match the capabilities correctly.

The next step is to create a Heat environment file (for example, **scheduler_hints_env.yaml**) that uses scheduler hints to match the capabilities for each node. For example:

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

To use these scheduler hints, include the ` scheduler_hints_env.yaml` environment file with the **overcloud deploy command** during Overcloud creation.

The same approach is possible for each role via these parameters:

- **ControllerSchedulerHints** for Controller nodes.
- **NovaComputeSchedulerHints** for Compute nodes.

- **`BlockStorageSchedulerHints`** for Block Storage nodes.
- **`ObjectStorageSchedulerHints`** for Object Storage nodes.
- **`CephStorageSchedulerHints`** for Ceph Storage nodes.
- **`[ROLE]SchedulerHints`** for custom roles. Replace **`[ROLE]`** with the role name.

NOTE

Node placement takes priority over profile matching. To avoid scheduling failures, use the default **`baremetal`** flavor for deployment and not the flavors designed for profile matching (**`compute`**, **`control`**, etc). For example:

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

10.2. ASSIGNING CUSTOM HOSTNAMES

In combination with the node ID configuration in Section 10.1, “Assigning Specific Node IDs”, the director can also assign a specific custom hostname to each node. This is useful when you need to define where a system is located (e.g. **`rack2-row12`**), match an inventory identifier, or other situations where a custom hostname is desired.

To customize node hostnames, use the **`HostnameMap`** parameter in an environment file, such as the ` scheduler_hints_env.yaml` file from Section 10.1, “Assigning Specific Node IDs”. For example:

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

Define the **`HostnameMap`** in the **`parameter_defaults`** section, and set each mapping as the original hostname that Heat defines using **`HostnameFormat`** parameters (e.g. **`overcloud-controller-0`**) and the second value is the desired custom hostname for that node (e.g. **`overcloud-controller-prod-123-0`**).

Using this method in combination with the node ID placement ensures each node has a custom hostname.

10.3. ASSIGNING PREDICTABLE IPS

For further control over the resulting environment, the director can assign Overcloud nodes with specific IPs on each network as well. Use the **`environments/ips-from-pool-all.yaml`** environment file in the core Heat template collection. Copy this file to the **`stack`** user’s **`templates`** directory.

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

There are two major sections in the **ips-from-pool-all.yaml** file.

The first is a set of **resource_registry** references that override the defaults. These tell the director to use a specific IP for a given port on a node type. Modify each resource to use the absolute path of its respective template. For example:

```
  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
```

The default configuration sets all networks on all node types to use pre-assigned IPs. To allow a particular network or node type to use default IP assignment instead, simply remove the **resource_registry** entries related to that node type or network from the environment file.

The second section is parameter_defaults, where the actual IP addresses are assigned. Each node type has an associated parameter:

- **ControllerIPs** for Controller nodes.
- **NovaComputeIPs** for Compute nodes.
- **CephStorageIPs** for Ceph Storage nodes.
- **BlockStorageIPs** for Block Storage nodes.
- **SwiftStorageIPs** for Object Storage nodes.
- **[ROLE]IPs** for custom roles. Replace **[ROLE]** with the role name.

Each parameter is a map of network names to a list of addresses. Each network type must have at least as many addresses as there will be nodes on that network. The director assigns addresses in order. The first node of each type receives the first address on each respective list, the second node receives the second address on each respective lists, and so forth.

For example, if an Overcloud will contain three Ceph Storage nodes, the CephStorageIPs parameter might look like:

```
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
```

The first Ceph Storage node receives two addresses: 172.16.1.100 and 172.16.3.100. The second receives 172.16.1.101 and 172.16.3.101, and the third receives 172.16.1.102 and 172.16.3.102. The same pattern applies to the other node types.

Make sure the chosen IP addresses fall outside the allocation pools for each network defined in your network environment file (see Section 8.2, "Creating a Network Environment File"). For example, make sure the **internal_api** assignments fall outside of the **InternalApiAllocationPools** range. This avoids conflicts with any IPs chosen automatically. Likewise, make sure the IP assignments do not conflict with the VIP configuration, either for standard predictable VIP placement (see Section 10.4, "Assigning Predictable Virtual IPs") or external load balancing (see Section 21.1, "Configuring External Load Balancing").

IMPORTANT

If an overcloud node is deleted, do not remove its entries in the IP lists. The IP list is based on the underlying Heat indices, which do not change even if you delete nodes. To indicate a given entry in the list is no longer used, replace the IP value with a value such as **DELETED** or **UNUSED**. Entries should never be removed from the IP lists, only changed or added.

To apply this configuration during a deployment, include the **ips-from-pool-all.yaml** environment file with the **openstack overcloud deploy** command.

IMPORTANT

If using network isolation (see Chapter 8, *Isolating Networks*), include the **ips-from-pool-all.yaml** file after the **network-isolation.yaml** file.

For example:

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

10.4. ASSIGNING PREDICTABLE VIRTUAL IPS

In addition to defining predictable IP addresses for each node, the director also provides a similar ability to define predictable Virtual IPs (VIPs) for clustered services. To accomplish this, edit the network environment file from Section 8.2, "Creating a Network Environment File" and add the VIP parameters in the **parameter_defaults** section:

```
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'}]
```

Select these IPs from outside of their respective allocation pool ranges. For example, select an IP address for **InternalApiVirtualFixedIPs** that is not within the **InternalApiAllocationPools** range.

This step is only for overclouds using the default internal load balancing configuration. If assigning VIPs with an external load balancer, use the procedure in the dedicated External Load Balancing for the Overcloud guide.

CHAPTER 11. ENABLING SSL/TLS ON OVERCLOUD PUBLIC ENDPOINTS

By default, the overcloud uses unencrypted endpoints for its services. This means that the overcloud configuration requires an additional environment file to enable SSL/TLS for its Public API endpoints. The following chapter shows how to configure your SSL/TLS certificate and include it as a part of your overcloud creation.

> **NOTE**
>
> This process only enables SSL/TLS for Public API endpoints. The Internal and Admin APIs remain unencrypted.

This process requires network isolation to define the endpoints for the Public API. See Chapter 8, *Isolating Networks* for instruction on network isolation.

11.1. INITIALIZING THE SIGNING HOST

The signing host is the host that generates new certificates and signs them with a certificate authority. If you have never created SSL certificates on the chosen signing host, you might need to initialize the host so that it can sign new certificates.

The **/etc/pki/CA/index.txt** file stores records of all signed certificates. Check if this file exists. If it does not exist, create an empty file:

```
$ sudo touch /etc/pki/CA/index.txt
```

The **/etc/pki/CA/serial** file identifies the next serial number to use for the next certificate to sign. Check if this file exists. If it does not exist, create a new file with a new starting value:

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

11.2. CREATING A CERTIFICATE AUTHORITY

Normally you sign your SSL/TLS certificates with an external certificate authority. In some situations, you might aim to use your own certificate authority. For example, you might aim to have an internal-only certificate authority.

For example, generate a key and certificate pair to act as the certificate authority:

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -
out ca.crt.pem
```

The **openssl req** command asks for certain details about your authority. Enter these details.

This creates a certificate authority file called **ca.crt.pem**.

11.3. ADDING THE CERTIFICATE AUTHORITY TO CLIENTS

For any external clients aiming to communicate using SSL/TLS, copy the certificate authority file to each client that requires access your Red Hat OpenStack Platform environment. Once copied to the client, run the following command on the client to add it to the certificate authority trust bundle:

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

For example, the undercloud requires a copy of the certificate authority file so that it can communicate with the overcloud endpoints during creation.

11.4. CREATING AN SSL/TLS KEY

Run the following commands to generate the SSL/TLS key (**server.key.pem**), which we use at different points to generate our undercloud or overcloud certificates:

```
$ openssl genrsa -out server.key.pem 2048
```

11.5. CREATING AN SSL/TLS CERTIFICATE SIGNING REQUEST

This next procedure creates a certificate signing request for the overcloud. Copy the default OpenSSL configuration file for customization.

```
$ cp /etc/pki/tls/openssl.cnf .
```

Edit the custom **openssl.cnf** file and set SSL parameters to use for the overcloud. An example of the types of parameters to modify include:

```
[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
```

Set the **commonName_default** to one of the following:

- If using an IP to access over SSL/TLS, use the Virtual IP for the Public API. Set this VIP using the **PublicVirtualFixedIPs** parameter in an environment file. For more information, see Section 10.4, “Assigning Predictable Virtual IPs”. If you are not using predictable VIPs, the director assigns the first IP address from the range defined in the **ExternalAllocationPools** parameter.

- If using a fully qualified domain name to access over SSL/TLS, use the domain name instead.

Include the same Public API IP address as an IP entry and a DNS entry in the **alt_names** section. If also using DNS, include the hostname for the server as DNS entries in the same section. For more information about **openssl.cnf**, run **man openssl.cnf**.

Run the following command to generate certificate signing request (**server.csr.pem**):

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out
server.csr.pem
```

Make sure to include the SSL/TLS key you created in Section 11.4, “Creating an SSL/TLS Key” for the **-key** option.

Use the **server.csr.pem** file to create the SSL/TLS certificate in the next section.

11.6. CREATING THE SSL/TLS CERTIFICATE

The following command creates a certificate for your undercloud or overcloud:

```
$ 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
```

This command uses:

- The configuration file specifying the v3 extensions. Include this as the **-config** option.

- The certificate signing request from Section 11.5, “Creating an SSL/TLS Certificate Signing Request” to generate the certificate and sign it throught a certificate authority. Include this as the **-in** option.

- The certificate authority you created in Section 11.2, “Creating a Certificate Authority”, which signs the certificate. Include this as the **-cert** option.

- The certificate authority private key you created in Section 11.2, “Creating a Certificate Authority”. Include this as the **-keyfile** option.

This results in a certificate named **server.crt.pem**. Use this certificate in conjunction with the SSL/TLS key from Section 11.4, “Creating an SSL/TLS Key” to enable SSL/TLS.

11.7. ENABLING SSL/TLS

Copy the **enable-tls.yaml** environment file from the Heat template collection:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/enable-tls.yaml ~/templates/.
```

Edit this file and make the following changes for these parameters:

SSLCertificate

Copy the contents of the certificate file (**server.crt.pem**) into the **SSLCertificate** parameter. For example:

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

IMPORTANT

The certificate contents require the same indentation level for all new lines.

SSLKey

Copy the contents of the private key (**server.key.pem**) into the **SSLKey** parameter. For example:

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO9hkJZnGP6qb6wtYUoy1bVP7
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4XpZUC7yhqPaU
    -----END RSA PRIVATE KEY-----
```

IMPORTANT

The private key contents require the same indentation level for all new lines.

OS::TripleO::NodeTLSData

Change the resource path for **OS::TripleO::NodeTLSData:** to an absolute path:

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

11.8. INJECTING A ROOT CERTIFICATE

If the certificate signer is not in the default trust store on the overcloud image, you must inject the certificate authority into the overcloud image. Copy the **inject-trust-anchor.yaml** environment file from the heat template collection:

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

Edit this file and make the following changes for these parameters:

SSLRootCertificate

Copy the contents of the root certificate authority file (**ca.crt.pem**) into the **SSLRootCertificate** parameter. For example:

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

IMPORTANT

The certificate authority contents require the same indentation level for all new lines.

OS::TripleO::NodeTLSCAData

Change the resource path for **OS::TripleO::NodeTLSCAData:** to an absolute path:

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

11.9. CONFIGURING DNS ENDPOINTS

If using a DNS hostname to access the overcloud through SSL/TLS, create a new environment file (**~/templates/cloudname.yaml**) to define the hostname of the overcloud's endpoints. Use the following parameters:

CloudName

The DNS hostname of the overcloud endpoints.

DnsServers

A list of DNS servers to use. The configured DNS servers must contain an entry for the configured **CloudName** that matches the IP address of the Public API.

An example of the contents for this file:

```
parameter_defaults:
  CloudName: overcloud.example.com
  DnsServers: ["10.0.0.254"]
```

11.10. ADDING ENVIRONMENT FILES DURING OVERCLOUD CREATION

The deployment command (**`openstack overcloud deploy`**) uses the **`-e`** option to add environment files. Add the environment files from this section in the following order:

- The environment file to enable SSL/TLS (**`enable-tls.yaml`**)
- The environment file to set the DNS hostname (**`cloudname.yaml`**)
- The environment file to inject the root certificate authority (**`inject-trust-anchor.yaml`**)
- The environment file to set the public endpoint mapping:
  - If using a DNS name for accessing the public endpoints, use **`/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-dns.yaml`**
  - If using a IP address for accessing the public endpoints, use **`/usr/share/openstack-tripleo-heat-templates/environments/tls-endpoints-public-ip.yaml`**

For example:

```
$ 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/tls-endpoints-public-dns.yaml
```

11.11. UPDATING SSL/TLS CERTIFICATES

If you need to update certificates in the future:

- Edit the **`enable-tls.yaml`** file and update the **`SSLCertificate`**, **`SSLKey`**, and **`SSLIntermediateCertificate`** parameters.
- If your certificate authority has changed, edit the **`inject-trust-anchor.yaml`** file and update the **`SSLRootCertificate`** parameter.

Once the new certificate content is in place, rerun your deployment command. For example:

```
$ 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/tls-endpoints-public-dns.yaml
```

CHAPTER 12. ENABLING SSL/TLS ON INTERNAL AND PUBLIC ENDPOINTS WITH IDENTITY MANAGEMENT

You can enable SSL/TLS on certain overcloud endpoints. Due to the number of certificates required, the director integrates with a Red Hat Identity Management (IdM) server to act as a certificate authority and manage the overcloud certificates. This process involves using **novajoin** to enroll overcloud nodes to the IdM server.

12.1. ADD THE UNDERCLOUD TO THE CA

Before deploying the overcloud, you must add the undercloud to the Certificate Authority (CA):

1. On the undercloud node, install the **python-novajoin** package:

   ```
   $ sudo yum install python-novajoin
   ```

2. On the undercloud node, run the **novajoin-ipa-setup** script, adjusting the values to suit your deployment:

   ```
   $ 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
   ```

   In the following section, you will use the resulting One-Time Password (OTP) to enroll the undercloud.

12.2. ADD THE UNDERCLOUD TO IDM

This procedure registers the undercloud with IdM and configures novajoin.

1. The novajoin service is disabled by default. To enable it, add an entry to **undercloud.conf**:

   ```
   enable_novajoin = true
   ```

2. You need set a One-Time Password (OTP) to register the undercloud node with IdM:

   ```
   ipa_otp = <otp>
   ```

3. Ensure the overcloud’s domain name served by neutron’s DHCP server matches the IdM domain (your kerberos realm in lowercase):

   ```
   overcloud_domain_name = <domain>
   ```

4. Set the appropriate hostname for the undercloud:

   ```
   undercloud_hostname = <undercloud FQDN>
   ```

5. Set IdM as the nameserver for the undercloud:

   ```
   undercloud_nameservers = <IdM IP>
   ```

6. For larger environments, you will need to review the novajoin connection timeout values. In **undercloud.conf**, add a reference to a new file called **undercloud-timeout.yaml**:

   ```
   hieradata_override = /home/stack/undercloud-timeout.yaml
   ```

   Add the following options to **undercloud-timeout.yaml**. You can specify the timeout value in seconds, for example, **5**:

   ```
   nova::api::vendordata_dynamic_connect_timeout: <timeout value>
   nova::api::vendordata_dynamic_read_timeout: <timeout value>
   ```

7. Save the **undercloud.conf** file.

8. Run the undercloud deployment command to apply the changes to your existing undercloud:

   ```
   $ openstack undercloud install
   ```

12.3. CONFIGURE OVERCLOUD DNS

For automatic detection of your IdM environment, and easier enrollment, consider using IdM as your DNS server:

1. Connect to your undercloud:

   ```
   $ source ~/stackrc
   ```

2. Configure the control plane subnet to use IdM as the DNS name server:

   ```
   $ openstack subnet set ctlplane-subnet --dns-nameserver
   <idm_server_address>
   ```

3. Set the **DnsServers** parameter in an environment file to use your IdM server:

   ```
   parameter_defaults:
     DnsServers: ["<idm_server_address>"]
   ```

   This parameter is usually defined in a custom **network-environment.yaml** file.

12.4. CONFIGURE OVERCLOUD TO USE NOVAJOIN

1. To enable IdM integration, create a copy of the **/usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml** environment file:

   ```
   $ cp /usr/share/openstack-tripleo-heat-
   templates/environments/predictable-placement/custom-domain.yaml \
     /home/stack/templates/custom-domain.yaml
   ```

2. Edit the **/home/stack/templates/custom-domain.yaml** environment file and set the **CloudDomain** and **CloudName*** values to suit your deployment. For example:

```
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. Include the following environment files in the overcloud deployment process:

   - **/usr/share/openstack-tripleo-heat-templates/environments/enable-internal-tls.yaml**

   - **/usr/share/openstack-tripleo-heat-templates/environments/tls-everywhere-endpoints-dns.yaml**

   - **/home/stack/templates/custom-domain.yaml**
     For example:

```
openstack overcloud deploy \
  --templates \
   -e /usr/share/openstack-tripleo-heat-
templates/environments/enable-internal-tls.yaml \
   -e /usr/share/openstack-tripleo-heat-
templates/environments/tls-everywhere-endpoints-dns.yaml \
   -e /home/stack/templates/custom-domain.yaml \
```

   As a result, the deployed overcloud nodes will be automatically enrolled with IdM.

4. This only sets TLS for the internal endpoints. For the external endpoints you can use the normal means of adding TLS with the **./tripleo-heat-templates/environments/enable-tls.yaml** environment file (which must be modified to add your custom certificate and key). Consequently, your **openstack deploy** command would be similar to this:

```
openstack overcloud deploy \
  --templates \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/tls-
everywhere-endpoints-dns.yaml \
  -e /home/stack/templates/custom-domain.yaml \
  -e /home/stack/templates/enable-tls.yaml
```

5. Alternatively, you can also use IdM to issue your public certificates. In that case, you need to use the **./tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml** environment file. For example:

```
openstack overcloud deploy \
  --templates \
   -e ./tripleo-heat-templates/environments/enable-internal-tls.yaml
\
   -e /usr/share/openstack-tripleo-heat-templates/environments/tls-
everywhere-endpoints-dns.yaml \
   -e /home/stack/templates/custom-domain.yaml \
   -e ./tripleo-heat-templates/environments/services/haproxy-public-
tls-certmonger.yaml
```

CHAPTER 13. DEBUG MODES

You can enable and disable the **DEBUG** level logging mode for certain services in the overcloud. To configure debug mode for a service, set the respective debug parameter. For example, OpenStack Identity (keystone) uses the **KeystoneDebug** parameter. Set this parameter in the **parameter_defaults** section of an environment file:

```
parameter_defaults:
  KeystoneDebug: True
```

For a full list of debug parameters, see "Debug Parameters" in the *Overcloud Parameters* guide.

CHAPTER 14. POLICIES

You can configure access policies for certain services in the overcloud. To configure policies for a service, set the respective policy parameter with a hash value containing the service's policies. For example, OpenStack Identity (keystone) uses the **`KeystonePolicies`** parameter. Set this parameter in the **`parameter_defaults`** section of an environment file:

```
parameter_defaults:
  KeystonePolicies: { keystone-context_is_admin: { key: context_is_admin,
value: 'role:admin' } }
```

For a full list of policy parameters, see "Policy Parameters" in the *Overcloud Parameters* guide.

CHAPTER 15. STORAGE CONFIGURATION

This chapter outlines several methods of configuring storage options for your Overcloud.

IMPORTANT

The Overcloud uses local and LVM storage for the default storage options. However, these options are not supported for enterprise-level Overclouds. It is recommended to use one of the storage options in this chapter.

15.1. CONFIGURING NFS STORAGE

This section describes configuring the Overcloud to use an NFS share. The installation and configuration process is based on the modification of an existing environment file in the core Heat template collection.

The core heat template collection contains a set of environment files in **`/usr/share/openstack-tripleo-heat-templates/environments/`**. These environment templates help with custom configuration of some of the supported features in a director-created Overcloud. This includes an environment file to help configure storage. This file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml`**. Copy this file to the **`stack`** user's template directory.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-
environment.yaml ~/templates/.
```

The environment file contains some parameters to help configure different storage options for OpenStack's block and image storage components, cinder and glance. In this example, you will configure the Overcloud to use an NFS share. Modify the following parameters:

CinderEnableIscsiBackend

Enables the iSCSI backend. Set to **`false`**.

CinderEnableRbdBackend

Enables the Ceph Storage backend. Set to **`false`**.

CinderEnableNfsBackend

Enables the NFS backend. Set to **`true`**.

NovaEnableRbdBackend

Enables Ceph Storage for Nova ephemeral storage. Set to **`false`**.

GlanceBackend

Define the back end to use for Glance. Set to **`file`** to use file-based storage for images. The Overcloud will save these files in a mounted NFS share for Glance.

CinderNfsMountOptions

The NFS mount options for the volume storage.

CinderNfsServers

The NFS share to mount for volume storage. For example, 192.168.122.1:/export/cinder.

GlanceNfsEnabled

Enables Pacemaker to manage the share for image storage. If disabled, the Overcloud stores images in the Controller node's file system. Set to **`true`**.

GlanceNfsShare

The NFS share to mount for image storage. For example, 192.168.122.1:/export/glance.

GlanceNfsOptions

The NFS mount options for the image storage.

The environment file's options should look similar to the following:

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
  GlanceBackend: 'file'

  CinderNfsMountOptions: 'rw,sync'
  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'
```

IMPORTANT

Include the **`context=system_u:object_r:glance_var_lib_t:s0`** in the **`GlanceNfsOptions`** parameter to allow glance access to the **`/var/lib`** directory. Without this SELinux content, glance will fail to write to the mount point.

These parameters are integrated as part of the heat template collection. Setting them as such creates two NFS mount points for cinder and glance to use.

Save this file for inclusion in the Overcloud creation.

15.2. CONFIGURING CEPH STORAGE

The director provides two main methods for integrating Red Hat Ceph Storage into an Overcloud.

Creating an Overcloud with its own Ceph Storage Cluster

The director has the ability to create a Ceph Storage Cluster during the creation on the Overcloud. The director creates a set of Ceph Storage nodes that use the Ceph OSD to store the data. In addition, the director install the Ceph Monitor service on the Overcloud's Controller nodes. This means if an organization creates an Overcloud with three highly available controller nodes, the Ceph Monitor also becomes a highly available service. For more information, see the Deploying an Overcloud with Containerized Red Hat Ceph guide.

Integrating a Existing Ceph Storage into an Overcloud

If you already have an existing Ceph Storage Cluster, you can integrate this during an Overcloud deployment. This means you manage and scale the cluster outside of the Overcloud configuration. For more information, see the Integrating an Overcloud with an Existing Red Hat Ceph Cluster guide.

15.3. USING AN EXTERNAL OBJECT STORAGE CLUSTER

You can reuse an external Object Storage (swift) cluster by disabling the default Object Storage service deployment on the controller nodes. Doing so disables both the proxy and storage services for Object Storage and configures haproxy and keystone to use the given external Swift endpoint.

> **NOTE**
>
> User accounts on the external Object Storage (swift) cluster have to be managed by hand.

You need the endpoint IP address of the external Object Storage cluster as well as the **`authtoken`** password from the external Object Storage **`proxy-server.conf`** file. You can find this information by using the **`openstack endpoint list`** command.

To deploy director with an external Swift cluster:

1. Create a new file named **`swift-external-params.yaml`** with the following content:

   - Replace **`EXTERNAL.IP:PORT`** with the IP address and port of the external proxy and

   - Replace **`AUTHTOKEN`** with the **`authtoken`** password for the external proxy on the **`SwiftPassword`** line.

   ```
   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. Save this file as **`swift-external-params.yaml`**.

3. Deploy the overcloud using these additional environment files.

   ```
   openstack overcloud deploy --templates \
   -e [your environment files]
   -e /usr/share/openstack-tripleo-heat-templates/environments/swift-
   external.yaml
   -e swift-external-params.yaml
   ```

15.4. CONFIGURING THIRD PARTY STORAGE

The director include a couple of environment files to help configure third-party storage providers. This includes:

Dell EMC Storage Center

Deploys a single Dell EMC Storage Center back end for the Block Storage (cinder) service. The environment file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellsc-config.yaml`**.

See the Dell Storage Center Back End Guide for full configuration information.

Dell EMC PS Series

Deploys a single Dell EMC PS Series back end for the Block Storage (cinder) service. The environment file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellps-config.yaml`**.

See the Dell EMC PS Series Back End Guide for full configuration information.

NetApp Block Storage

Deploys a NetApp storage appliance as a back end for the Block Storage (cinder) service. The environment file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/cinder-netapp-config.yaml`**.

See the NetApp Block Storage Back End Guide for full configuration information.

CHAPTER 16. SECURITY ENHANCEMENTS

The following sections provide some suggestions to harden the security of your overcloud.

16.1. MANAGING THE OVERCLOUD FIREWALL

Each of the core OpenStack Platform services contains firewall rules in their respective composable service templates. This automatically creates a default set of firewall rules for each overcloud node.

The overcloud Heat templates contain a set of parameters to help with additional firewall management:

ManageFirewall

Defines whether to automatically manage the firewall rules. Set to `true` to allow Puppet to automatically configure the firewall on each node. Set to `false` if you want to manually manage the firewall. The default is `true`.

PurgeFirewallRules

Defines whether to purge the default Linux firewall rules before configuring new ones. The default is `false`.

If `ManageFirewall` is set to `true`, you can create additional firewall rules on deployment. Set the `tripleo::firewall::firewall_rules` hieradata using a configuration hook (see Section 4.5, “Puppet: Customizing Hieradata for Roles”) in an environment file for your overcloud. This hieradata is a hash containing the firewall rule names and their respective parameters as keys, all of which are optional:

port

The port associated to the rule.

dport

The destination port associated to the rule.

sport

The source port associated to the rule.

proto

The protocol associated to the rule. Defaults to `tcp`.

action

The action policy associated to the rule. Defaults to `accept`.

jump

The chain to jump to. If present, it overrides `action`.

state

An Array of states associated to the rule. Defaults to `['NEW']`.

source

The source IP address associated to the rule.

iniface

The network interface associated to the rule.

chain

The chain associated to the rule. Defaults to `INPUT`.

destination

The destination CIDR associated to the rule.

The following example demonstrates the syntax of the firewall rule format:

```
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
```

This applies two additional firewall rules to all nodes through `ExtraConfig`.

NOTE

Each rule name becomes the comment for the respective `iptables` rule. Note also each rule name starts with a three-digit prefix to help Puppet order all defined rules in the final `iptables` file. The default OpenStack Platform rules use prefixes in the 000 to 200 range.

16.2. CHANGING THE SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) STRINGS

The director provides a default read-only SNMP configuration for your overcloud. It is advisable to change the SNMP strings to mitigate the risk of unauthorized users learning about your network devices.

Set the following hieradata using the `ExtraConfig` hook in an environment file for your overcloud:

snmp::ro_community

IPv4 read-only SNMP community string. The default value is `public`.

snmp::ro_community6

IPv6 read-only SNMP community string. The default value is `public`.

snmp::ro_network

Network that is allowed to `RO query` the daemon. This value can be a string or an array. Default value is `127.0.0.1`.

snmp::ro_network6

Network that is allowed to `RO query` the daemon with IPv6. This value can be a string or an array. The default value is `::1/128`.

snmp::snmpd_config

Array of lines to add to the ***snmpd.conf*** file as a safety valve. The default value is `[]`. See the SNMP Configuration File web page for all available options.

For example:

```
parameter_defaults:
  ExtraConfig:
    snmp::ro_community: mysecurestring
    snmp::ro_community6: myv6securestring
```

This changes the read-only SNMP community string on all nodes.

16.3. CHANGING THE SSL/TLS CIPHER AND RULES FOR HAPROXY

If you enabled SSL/TLS in the overcloud (see Chapter 11, *Enabling SSL/TLS on Overcloud Public Endpoints*), you might want to harden the SSL/TLS ciphers and rules used with the HAProxy configuration. This helps avoid SSL/TLS vulnerabilities, such as the POODLE vulnerability.

Set the following hieradata using the **`ExtraConfig`** hook in an environment file for your overcloud:

tripleo::haproxy::ssl_cipher_suite

The cipher suite to use in HAProxy.

tripleo::haproxy::ssl_options

The SSL/TLS rules to use in HAProxy.

For example, you might aim to use the following cipher and rules:

- Cipher: **`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`**

- Rules: **`no-sslv3 no-tls-tickets`**

Create an environment file with the following content:

```
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
```

NOTE

The cipher collection is one continuous line.

Include this environment file with your overcloud creation.

16.4. USING THE OPEN VSWITCH FIREWALL

You can configure security groups to use the Open vSwitch (OVS) firewall driver in Red Hat OpenStack Platform director. The **NeutronOVSFirewallDriver** parameter allows you to specify which firewall driver to use:

- **iptables_hybrid** - Configures neutron to use the iptables/hybrid based implementation.
- **openvswitch** - Configures neutron to use the OVS firewall flow-based driver.

The **openvswitch** firewall driver includes higher performance and reduces the number of interfaces and bridges used to connect guests to the project network.

NOTE

The **iptables_hybrid** option is not compatible with OVS-DPDK.

Configure the **NeutronOVSFirewallDriver** parameter in the **network-environment.yaml** file:

```
NeutronOVSFirewallDriver: openvswitch
```

- **NeutronOVSFirewallDriver** : Configures the name of the firewall driver to use when implementing security groups. Possible values depend on your system configuration; some examples are: **noop**, **openvswitch**, **iptables_hybrid**. The default value of an empty string results in a supported configuration.

16.5. USING SECURE ROOT USER ACCESS

The overcloud image automatically contains hardened security for the **root** user. For example, each deployed overcloud node automatically disables direct SSH access to the **root** user. You can still access the **root** user on overcloud nodes through the following method:

1. Log into the undercloud node’s **stack** user.
2. Each overcloud node has a **heat-admin** user account. This user account contains the undercloud’s public SSH key, which provides SSH access without a password from the undercloud to the overcloud node. On the undercloud node, log into the chosen overcloud node through SSH using the **heat-admin** user.
3. Switch to the **root** user with **sudo -i**.

Reducing Root User Security

Some situations might require direct SSH access to the **root** user. In this case, you can reduce the SSH restrictions on the **root** user for each overcloud node.

WARNING

This method is intended for debugging purposes only. It is not recommended for use in a production environment.

The method uses the first boot configuration hook (see Section 4.1, “First Boot: Customizing First Boot Configuration”). Place the following content in an environment file:

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-templates/firstboot/userdata_root_password.yaml

parameter_defaults:
  NodeRootPassword: "p@55w0rd!"
```

Note the following:

- The **`OS::TripleO::NodeUserData`** resource refers to the a template that configures the **`root`** user during the first boot **`cloud-init`** stage.

- The **`NodeRootPassword`** parameter sets the password for the **`root`** user. Change the value of this parameter to your desired password. Note the environment file contains the password as a plain text string, which is considered a security risk.

Include this environment file with the **`openstack overcloud deploy`** command when creating your overcloud.

CHAPTER 17. FENCING THE CONTROLLER NODES

Fencing is the process of isolating a failed node to protect a cluster and its resources. Without fencing, a failed node can result in data corruption in a cluster.

The director uses Pacemaker to provide a highly available cluster of Controller nodes. Pacemaker uses a process called STONITH to fence failed nodes. STONITH is disabled by default and requires manual configuration so that Pacemaker can control the power management of each node in the cluster.

17.1. REVIEW THE PREREQUISITES

To configure fencing in the overcloud, your overcloud must already have been deployed and be in a working state. The following steps review the state of Pacemaker and STONITH in your deployment:

1. Log in to each node as the **heat-admin** user from the **stack** user on the director. The overcloud creation automatically copies the **stack** user's SSH key to each node's **heat-admin**.

2. Verify you have a running cluster:

   ```
   $ sudo pcs status
   Cluster name: openstackHA
   Last updated: Wed Jun 24 12:40:27 2015
   Last change: Wed Jun 24 11:36:18 2015
   Stack: corosync
   Current DC: lb-c1a2 (2) - partition with quorum
   Version: 1.1.12-a14efad
   3 Nodes configured
   141 Resources configured
   ```

3. Verify STONITH is disabled:

   ```
   $ sudo pcs property show
   Cluster Properties:
   cluster-infrastructure: corosync
   cluster-name: openstackHA
   dc-version: 1.1.12-a14efad
   have-watchdog: false
   stonith-enabled: false
   ```

17.2. ENABLE FENCING

Having confirmed your overcloud is deployed and working, you can then configure fencing:

1. Generate the **fencing.yaml** file:

   ```
   $ openstack overcloud generate fencing --ipmi-lanplus --ipmi-level
   administrator --output fencing.yaml instackenv.json
   ```

   - Sample **fencing.yaml** file:

     ```
     parameter_defaults:
       EnableFencing: true
     ```

```
  FencingConfig:
    devices:
    - agent: fence_ipmilan
      host_mac: 11:11:11:11:11:11
      params:
        ipaddr: 10.0.0.101
        lanplus: true
        login: admin
        passwd: InsertComplexPasswordHere
        pcmk_host_list: host04
        privlvl: administrator
```

2. Pass the resulting **fencing.yaml** file to the **deploy** command you previously used to deploy the overcloud. This will re-run the deployment procedure and configure fencing on the hosts:

   ```
   openstack overcloud deploy --templates -e /usr/share/openstack-
   tripleo-heat-templates/environments/network-isolation.yaml -e
   ~/templates/network-environment.yaml -e ~/templates/storage-
   environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-
   scale 3 --control-flavor control --compute-flavor compute --ceph-
   storage-flavor ceph-storage --ntp-server pool.ntp.org --neutron-
   network-type vxlan --neutron-tunnel-types vxlan -e fencing.yaml
   ```

   The deployment command should complete without any error or exceptions.

3. Log in to the overcloud and verify fencing was configured for each of the controllers:

   a. Check the fencing resources are managed by Pacemaker:

   ```
   $ source stackrc
   $ nova list | grep controller
   $ ssh heat-admin@<controller-x_ip>
   $ sudo pcs status |grep fence
   stonith-overcloud-controller-x (stonith:fence_ipmilan): Started
   overcloud-controller-y
   ```

   You should see Pacemaker is configured to use a STONITH resource for each of the controllers specified in **fencing.yaml**. The **fence-resource** process should not be configured on the same host it controls.

   b. Use **pcs** to verify the fence resource attributes:

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

   The values used by STONITH should match those defined in the **fencing.yaml**.

17.3. TEST FENCING

This procedure tests whether fencing is working as expected.

1. Trigger a fencing action for each controller in the deployment:

   a. Log in to a controller:

```
$ source stackrc
$ nova list |grep controller
$ ssh heat-admin@<controller-x_ip>
```

b. As root, trigger fencing by using **`iptables`** to close all ports:

```
$ sudo -i
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
&&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -
j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 5016
-j ACCEPT &&
iptables -A INPUT -p udp -m state --state NEW -m udp --dport 5016
-j ACCEPT &&
iptables -A INPUT ! -i lo -j REJECT --reject-with icmp-host-
prohibited &&
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT &&
iptables -A OUTPUT -p tcp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT -p udp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT ! -o lo -j REJECT --reject-with icmp-host-
prohibited
```

As a result, the connections should drop, and the server should be rebooted.

c. From another controller, locate the fencing event in the Pacemaker log file:

```
$ ssh heat-admin@<controller-x_ip>
$ less /var/log/cluster/corosync.log
(less): /fenc*
```

You should see that STONITH has issued a fence action against the controller, and that Pacemaker has raised an event in the log.

d. Verify the rebooted controller has returned to the cluster:

   i. From the second controller, wait a few minutes and run **`pcs status`** to see if the fenced controller has returned to the cluster. The duration can vary depending on your configuration.

CHAPTER 18. CONFIGURING MONITORING TOOLS

Monitoring tools are an optional suite of tools that can be used for availability monitoring and centralized logging. The availability monitoring allows you to monitor the functionality of all components, while the centralized logging allows you to view all of the logs across your OpenStack environment in one central place.

For more information about configuring monitoring tools, see the dedicated Monitoring Tools Configuration Guide for full instructions.

CHAPTER 19. CONFIGURING NETWORK PLUGINS

The director includes environment files to help configure third-party network plugins:

19.1. FUJITSU CONVERGED FABRIC (C-FABRIC)

You can enable the Fujitsu Converged Fabric (C-Fabric) plugin using the environment file located at **`/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml`**.

1. Copy the environment file to your **`templates`** subdirectory:

   ```
   $ cp /usr/share/openstack-tripleo-heat-
   templates/environments/neutron-ml2-fujitsu-cfab.yaml
   /home/stack/templates/
   ```

2. Edit the **`resource_registry`** to use an absolute path:

   ```
   resource_registry:
     OS::TripleO::Services::NeutronML2FujitsuCfab:
   /usr/share/openstack-tripleo-heat-templates/puppet/services/neutron-
   plugin-ml2-fujitsu-cfab.yaml
   ```

3. Review the **`parameter_defaults`** in **`/home/stack/templates/neutron-ml2-fujitsu-cfab.yaml`**:

   - **`NeutronFujitsuCfabAddress`** - The telnet IP address of the C-Fabric. (string)
   - **`NeutronFujitsuCfabUserName`** - The C-Fabric username to use. (string)
   - **`NeutronFujitsuCfabPassword`** - The password of the C-Fabric user account. (string)
   - **`NeutronFujitsuCfabPhysicalNetworks`** - List of **`<physical_network>:<vfab_id>`** tuples that specify **`physical_network`** names and their corresponding vfab IDs. (comma_delimited_list)
   - **`NeutronFujitsuCfabSharePprofile`** - Determines whether to share a C-Fabric pprofile among neutron ports that use the same VLAN ID. (boolean)
   - **`NeutronFujitsuCfabPprofilePrefix`** - The prefix string for pprofile name. (string)
   - **`NeutronFujitsuCfabSaveConfig`** - Determines whether to save the configuration. (boolean)

4. To apply the template to your deployment, include the environment file in the **`openstack overcloud deploy`** command. For example:

   ```
   $ openstack overcloud deploy --templates -e
   /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml [OTHER OPTIONS]
   ...
   ```

19.2. FUJITSU FOS SWITCH

You can enable the Fujitsu FOS Switch plugin using the environment file located at **/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml**.

1. Copy the environment file to your **templates** subdirectory:

   ```
   $ cp /usr/share/openstack-tripleo-heat-
   templates/environments/neutron-ml2-fujitsu-fossw.yaml
   /home/stack/templates/
   ```

2. Edit the **resource_registry** to use an absolute path:

   ```
   resource_registry:
     OS::TripleO::Services::NeutronML2FujitsuFossw:
   /usr/share/openstack-tripleo-heat-templates/puppet/services/neutron-
   plugin-ml2-fujitsu-fossw.yaml
   ```

3. Review the **parameter_defaults** in **/home/stack/templates/neutron-ml2-fujitsu-fossw.yaml**:

   - **NeutronFujitsuFosswIps** - The IP addresses of all FOS switches. (comma_delimited_list)
   - **NeutronFujitsuFosswUserName** - The FOS username to use. (string)
   - **NeutronFujitsuFosswPassword** - The password of the FOS user account. (string)
   - **NeutronFujitsuFosswPort** - The port number to use for the SSH connection. (number)
   - **NeutronFujitsuFosswTimeout** - The timeout period of the SSH connection. (number)
   - **NeutronFujitsuFosswUdpDestPort** - The port number of the VXLAN UDP destination on the FOS switches. (number)
   - **NeutronFujitsuFosswOvsdbVlanidRangeMin** - The minimum VLAN ID in the range that is used for binding VNI and physical port. (number)
   - **NeutronFujitsuFosswOvsdbPort** - The port number for the OVSDB server on the FOS switches. (number)

4. To apply the template to your deployment, include the environment file in the **openstack overcloud deploy** command. For example:

   ```
   $ openstack overcloud deploy --templates -e
   /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS]
   ...
   ```

CHAPTER 20. CONFIGURING IDENTITY

The director includes parameters to help configure Identity Service (keystone) settings:

20.1. REGION NAME

By default, your overcloud's region will be named **`regionOne`**. You can change this by adding a **`KeystoneRegion`** entry your environment file. This setting cannot be changed post-deployment:

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

CHAPTER 21. OTHER CONFIGURATIONS

21.1. CONFIGURING EXTERNAL LOAD BALANCING

An Overcloud uses multiple Controllers together as a high availability cluster, which ensures maximum operational performance for your OpenStack services. In addition, the cluster provides load balancing for access to the OpenStack services, which evenly distributes traffic to the Controller nodes and reduces server overload for each node. It is also possible to use an external load balancer to perform this distribution. For example, an organization might use their own hardware-based load balancer to handle traffic distribution to the Controller nodes.

For more information about configuring external load balancing, see the dedicated External Load Balancing for the Overcloud guide for full instructions.

21.2. CONFIGURING IPV6 NETWORKING

As a default, the Overcloud uses Internet Protocol version 4 (IPv4) to configure the service endpoints. However, the Overcloud also supports Internet Protocol version 6 (IPv6) endpoints, which is useful for organizations that support IPv6 infrastructure. The director includes a set of environment files to help with creating IPv6-based Overclouds.

For more information about configuring IPv6 in the Overcloud, see the dedicated IPv6 Networking for the Overcloud guide for full instructions.

APPENDIX A. NETWORK ENVIRONMENT OPTIONS

Table A.1. Network Environment Options

Parameter	Description	Example
InternalApiNetCidr	The network and subnet for the Internal API network	172.17.0.0/24
StorageNetCidr	The network and subnet for the Storage network	
StorageMgmtNetCidr	The network and subnet for the Storage Management network	
TenantNetCidr	The network and subnet for the Tenant network	
ExternalNetCidr	The network and subnet for the External network	
InternalApiAllocationPools	The allocation pool for the Internal API network in a tuple format	[{*start*: *172.17.0.10*, *end*: *172.17.0.200*}]
StorageAllocationPools	The allocation pool for the Storage network in a tuple format	
StorageMgmtAllocationPools	The allocation pool for the Storage Management network in a tuple format	
TenantAllocationPools	The allocation pool for the Tenant network in a tuple format	
ExternalAllocationPools	The allocation pool for the External network in a tuple format	
InternalApiNetworkVlanID	The VLAN ID for the Internal API network	200
StorageNetworkVlanID	The VLAN ID for the Storage network	
StorageMgmtNetworkVlanID	The VLAN ID for the Storage Management network	
TenantNetworkVlanID	The VLAN ID for the Tenant network	

Parameter	Description	Example
ExternalNetworkVlanID	The VLAN ID for the External network	
ExternalInterfaceDefaultRoute	The gateway IP address for the External network	10.1.2.1
ControlPlaneDefaultRoute	Gateway router for the Provisioning network (or Undercloud IP)	ControlPlaneDefaultRoute: 192.0.2.254
ControlPlaneSubnetCidr	CIDR subnet mask length for provisioning network	ControlPlaneSubnetCidr: 24
EC2MetadataIp	The IP address of the EC2 metadata server. Generally the IP of the Undercloud.	EC2MetadataIp: 192.0.2.1
DnsServers	Define the DNS servers for the Overcloud nodes. Include a maximum of two.	DnsServers: ["8.8.8.8","8.8.4.4"]
BondInterfaceOvsOptions	The options for bonding interfaces	BondInterfaceOvsOptions:"bond_mode=balance-slb"
NeutronFlatNetworks	Defines the flat networks to configure in neutron plugins. Defaults to "datacentre" to permit external network creation	NeutronFlatNetworks: "datacentre"
NeutronExternalNetworkBridge	An Open vSwitch bridge to create on each hypervisor. Typically, this should not need to be changed.	NeutronExternalNetworkBridge: "''"
NeutronBridgeMappings	The logical to physical bridge mappings to use. Defaults to mapping the external bridge on hosts (br-ex) to a physical name (datacentre). You would use this for the default floating network	NeutronBridgeMappings: "datacentre:br-ex"
NeutronPublicInterface	Defines the interface to bridge onto br-ex for network nodes	NeutronPublicInterface: "eth0"
NeutronNetworkType	The tenant network type for Neutron	NeutronNetworkType: "vxlan"

Parameter	Description	Example
NeutronTunnelTypes	The tunnel types for the neutron tenant network. To specify multiple values, use a comma separated string.	NeutronTunnelTypes: *gre,vxlan*
NeutronTunnelIdRanges	Ranges of GRE tunnel IDs to make available for tenant network allocation	NeutronTunnelIdRanges "1:1000"
NeutronVniRanges	Ranges of VXLAN VNI IDs to make available for tenant network allocation	NeutronVniRanges: "1:1000"
NeutronEnableTunnelling	Defines whether to enable or disable tunneling in case you aim to use a VLAN segmented network or flat network with Neutron. Defaults to enabled	
NeutronNetworkVLANRanges	The neutron ML2 and Open vSwitch VLAN mapping range to support. Defaults to permitting any VLAN on the *datacentre* physical network.	NeutronNetworkVLANRanges: "datacentre:1:1000"
NeutronMechanismDrivers	The mechanism drivers for the neutron tenant network. Defaults to "openvswitch". To specify multiple values, use a comma-separated string	NeutronMechanismDrivers: *openvswitch,l2population*

APPENDIX B. NETWORK INTERFACE TEMPLATE EXAMPLES

This appendix provides a few example Heat templates to demonstrate network interface configuration.

B.1. CONFIGURING INTERFACES

Individual interfaces might require modification. The example below shows modifications required to use the second NIC to connect to an infrastructure network with DHCP addresses, and to use the third and fourth NICs for the bond:

```
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
```

The network interface template uses either the actual interface name ("eth0", "eth1", "enp0s25") or a set of numbered interfaces ("nic1", "nic2", "nic3"). The network interfaces of hosts within a role do not have to be exactly the same when using numbered interfaces (**nic1**, **nic2**, etc.) instead of named interfaces (**eth0**, **eno2**, etc.). For example, one host might have interfaces **em1** and **em2**, while another has **eno1** and **eno2**, but you can refer to both hosts' NICs as **nic1** and **nic2**.

The order of numbered interfaces corresponds to the order of named network interface types:

- **ethX** interfaces, such as **eth0**, **eth1**, etc. These are usually onboard interfaces.
- **enoX** interfaces, such as **eno0**, **eno1**, etc. These are usually onboard interfaces.
- **enX** interfaces, sorted alpha numerically, such as **enp3s0**, **enp3s1**, **ens3**, etc. These are usually add-on interfaces.

The numbered NIC scheme only takes into account the interfaces that are live, for example, if they have a cable attached to the switch. If you have some hosts with four interfaces and some with six interfaces, you should use **nic1** to **nic4** and only plug four cables on each host.

B.2. CONFIGURING ROUTES AND DEFAULT ROUTES

There are two ways a host has default routes set. If the interface is using DHCP and the DHCP server offers a gateway address, the system uses a default route for that gateway. Otherwise, you can set a default route on an interface with a static IP.

Although the Linux kernel supports multiple default gateways, it only uses the one with the lowest metric. If there are multiple DHCP interfaces, this can result in an unpredictable default gateway. In this case, it is recommended to set **defroute=no** for interfaces other than the one using the default route.

For example, you might want a DHCP interface (**nic3**) to be the default route. Use the following YAML to disable the default route on another DHCP interface (**nic2**):

```
# 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
```

NOTE

The **defroute** parameter only applies to routes obtained through DHCP.

To set a static route on an interface with a static IP, specify a route to the subnet. For example, you can set a route to the 10.1.2.0/24 subnet through the gateway at 172.17.0.1 on the Internal API network:

```
    - 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
```

B.3. CONFIGURE INTERFACE MAPPING

Director uses aliases such as **nic1** or **nic2** when referring to physical network interfaces. You can use **interface_mapping** to hardcode physical interfaces to specific aliases. This allows you to be predetermine which physical NIC will be mapped as **nic1** or **nic2** and so on. You can also map a MAC address to a specified alias.

NOTE

Normally, **os-net-config** will only register interfaces that are already connected in an **UP** state. However, if you do hardcode interfaces using a custom mapping file, then the interface is registered even if it is in a **DOWN** state.

Interfaces are mapped to aliases using an environment file. In this example, each node has predefined entries for **nic1** and **nic2**:

```
    parameter_defaults:
```

```
NetConfigDataLookup:
  node1:
    nic1: "em1"
    nic2: "em2"
  node2:
    nic1: "00:50:56:2F:9F:2E"
    nic2: "em2"
```

The resulting configuration is then applied by **`os-net-config`**. On each node, you can see the applied configuration under **`interface_mapping`** in **`/etc/os-net-config/mapping.yaml`**.

B.4. USING THE NATIVE VLAN FOR FLOATING IPS

Neutron uses a default empty string for its external bridge mapping. This maps the physical interface to the **`br-int`** instead of using **`br-ex`** directly. This model allows multiple Floating IP networks using either VLANs or multiple physical connections.

Use the **`NeutronExternalNetworkBridge`** parameter in the **`parameter_defaults`** section of your network isolation environment file:

```
parameter_defaults:
  # Set to "br-ex" when using floating IPs on the native VLAN
  NeutronExternalNetworkBridge: "''"
```

Using only one Floating IP network on the native VLAN of a bridge means you can optionally set the neutron external bridge. This results in the packets only having to traverse one bridge instead of two, which might result in slightly lower CPU usage when passing traffic over the Floating IP network.

B.5. USING THE NATIVE VLAN ON A TRUNKED INTERFACE

If a trunked interface or bond has a network on the native VLAN, the IP addresses are assigned directly to the bridge and there will be no VLAN interface.

For example, if the External network is on the native VLAN, a bonded configuration looks like this:

```
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
```

NOTE

When moving the address (and possibly route) statements onto the bridge, remove the corresponding VLAN interface from the bridge. Make the changes to all applicable roles. The External network is only on the controllers, so only the controller template requires a change. The Storage network on the other hand is attached to all roles, so if the Storage network is on the default VLAN, all roles require modifications.

B.6. CONFIGURING JUMBO FRAMES

The Maximum Transmission Unit (MTU) setting determines the maximum amount of data transmitted with a single Ethernet frame. Using a larger value results in less overhead since each frame adds data in the form of a header. The default value is 1500 and using a higher value requires the configuration of the switch port to support jumbo frames. Most switches support an MTU of at least 9000, but many are configured for 1500 by default.

The MTU of a VLAN cannot exceed the MTU of the physical interface. Make sure to include the MTU value on the bond and/or interface.

The Storage, Storage Management, Internal API, and Tenant networking all benefit from jumbo frames. In testing, Tenant networking throughput was over 300% greater when using jumbo frames in conjunction with VXLAN tunnels.

NOTE

It is recommended that the Provisioning interface, External interface, and any floating IP interfaces be left at the default MTU of 1500. Connectivity problems are likely to occur otherwise. This is because routers typically cannot forward jumbo frames across Layer 3 boundaries.

```
- 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
```

CHAPTER 22. NETWORK INTERFACE PARAMETERS

The following tables define the Heat template parameters for network interface types.

22.1. INTERFACE OPTIONS

Option	Default	Description
name		Name of the Interface
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the interface
routes		A sequence of routes assigned to the interface
mtu	1500	The maximum transmission unit (MTU) of the connection
primary	False	Defines the interface as the primary interface
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the interface

22.2. VLAN OPTIONS

Option	Default	Description
vlan_id		The VLAN ID

device		The VLAN’s parent device to attach the VLAN. For example, use this parameter to attach the VLAN to a bonded interface device.
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the VLAN
routes		A sequence of routes assigned to the VLAN
mtu	1500	The maximum transmission unit (MTU) of the connection
primary	False	Defines the VLAN as the primary interface
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the VLAN

22.3. OVS BOND OPTIONS

Option	Default	Description
name		Name of the bond
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the bond

routes		A sequence of routes assigned to the bond
mtu	1500	The maximum transmission unit (MTU) of the connection
primary	False	Defines the interface as the primary interface
members		A sequence of interface objects to use in the bond
ovs_options		A set of options to pass to OVS when creating the bond
ovs_extra		A set of options to to set as the OVS_EXTRA parameter in the bond’s network configuration file
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bond

22.4. OVS BRIDGE OPTIONS

Option	Default	Description
name		Name of the bridge
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the bridge
routes		A sequence of routes assigned to the bridge

mtu	1500	The maximum transmission unit (MTU) of the connection
members		A sequence of interface, VLAN, and bond objects to use in the bridge
ovs_options		A set of options to pass to OVS when creating the bridge
ovs_extra		A set of options to to set as the OVS_EXTRA parameter in the bridge’s network configuration file
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bridge

22.5. LINUX BOND OPTIONS

Option	Default	Description
name		Name of the bond
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the bond
routes		A sequence of routes assigned to the bond
mtu	1500	The maximum transmission unit (MTU) of the connection

primary	False	Defines the interface as the primary interface
members		A sequence of interface objects to use in the bond
bonding_options		A set of options when creating the bond. For more information on Linux bonding options, see 4.5.1. Bonding Module Directives in the Red Hat Enterprise Linux 7 Networking Guide.
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bond

22.6. LINUX BRIDGE OPTIONS

Option	Default	Description
name		Name of the bridge
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the bridge
routes		A sequence of routes assigned to the bridge
mtu	1500	The maximum transmission unit (MTU) of the connection
members		A sequence of interface, VLAN, and bond objects to use in the bridge

defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bridge

APPENDIX C. OPEN VSWITCH BONDING OPTIONS

The Overcloud provides networking through Open vSwitch (OVS), which provides several options for bonded interfaces. In Section 8.2, “Creating a Network Environment File”, you can configure a bonded interface in the network environment file using the following parameter:

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

C.1. CHOOSING A BOND MODE

By default, you cannot use LACP with OVS-based bonds. This configuration is not supported due to a known issue with some versions of Open vSwitch. Instead, consider using *bond_mode=balance-slb* as a replacement for this functionality. In addition, you can still use LACP with Linux bonding in your network interface templates. For example:

```
      - 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"
```

- **`mode`** - enables LACP.
- **`lacp_rate`** - defines whether LACP packets are sent every 1 second, or every 30 seconds.
- **`updelay`** - defines the minimum amount of time that an interface must be active before it is used for traffic (this helps mitigate port flapping outages).
- **`miimon`** - the interval in milliseconds that is used for monitoring the port state using the driver’s MIIMON functionality.

If you still want to use LACP with OVS-base bonds, you can manually delete the following lines from each network interface configuration (NIC) file before deployment:

```
    constraints:
      - allowed_pattern: "^((?!balance.tcp).)*$"
        description: |
          The balance-tcp bond mode is known to cause packet loss and
          should not be used in BondInterfaceOvsOptions.
```

After you delete the constraint from each NIC file, you can set the bond mode option in the bond interface parameter:

```
  BondInterfaceOvsOptions:
    "bond_mode=balance-tcp"
```

For the technical details behind this constraint, see BZ#1267291.

For more information on Linux bonding options, see 4.5.1. Bonding Module Directives in the *Red Hat Enterprise Linux 7 Networking Guide*.

C.2. BONDING OPTIONS

The following table provides some explanation of these options and some alternatives depending on your hardware.

Table C.1. Bonding Options

`bond_mode=balance-slb`	Balances flows based on source MAC address and output VLAN, with periodic rebalancing as traffic patterns change. Bonding with **`balance-slb`** allows a limited form of load balancing without the remote switch's knowledge or cooperation. SLB assigns each source MAC and VLAN pair to a link and transmits all packets from that MAC and VLAN through that link. This mode uses a simple hashing algorithm based on source MAC address and VLAN number, with periodic rebalancing as traffic patterns change. This mode is similar to mode 2 bonds used by the Linux bonding driver. This mode is used when the switch is configured with bonding but is not configured to use LACP (static instead of dynamic bonds).
`bond_mode=active-backup`	This mode offers active/standby failover where the standby NIC resumes network operations when the active connection fails. Only one MAC address is presented to the physical switch. This mode does not require any special switch support or configuration, and works when the links are connected to separate switches. This mode does not provide load balancing.
`lacp=[active|passive|off]`	Controls the Link Aggregation Control Protocol (LACP) behavior. Only certain switches support LACP. If your switch does not support LACP, use **`bond_mode=balance-slb`** or **`bond_mode=active-backup`**.
`other-config:lacp-fallback-ab=true`	Sets the LACP behavior to switch to bond_mode=active-backup as a fallback.
`other_config:lacp-time=[fast|slow]`	Set the LACP heartbeat to 1 second (fast) or 30 seconds (slow). The default is slow.
`other_config:bond-detect-mode=[miimon|carrier]`	Set the link detection to use miimon heartbeats (miimon) or monitor carrier (carrier). The default is carrier.
`other_config:bond-miimon-interval=100`	If using miimon, set the heartbeat interval in milliseconds.

`bond_updelay=1000`	Number of milliseconds a link must be up to be activated to prevent flapping.
`other_config:bond-rebalance-interval=10000`	Milliseconds between rebalancing flows between bond members. Set to zero to disable.

IMPORTANT

If you experience packet drops or performance issues using Linux bonds with Provider networks, consider disabling Large Receive Offload (LRO) on the standby interfaces. Avoid adding a Linux bond to an OVS bond, as port-flapping and loss of connectivity can occur. This is a result of a packet-loop through the standby interface.